conference

proceedings

2nd Large Installation System Administration of Windows NT Conference

Seattle, Washington, USA
July 14–17, 1999

Co-Sponsored by

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SAGE
THE SYSTEM ADMINISTRATORS GUILD

USENIX Proceedings of the 2nd Large Installation System Administration of Windows NT Conference Seattle, Washington, USA, July 1999

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
WWW URL: http://www.usenix.org

The price is $18 for members and $24 for nonmembers.
Outside the U.S.A. and Canada, please add
$10 per copy for postage (via air printed matter).

Past LISA-NT Proceedings

LISA-NT	1998	Seattle, Washington, USA	$18/24

ISBN 1-880446-30-8

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the

2nd Large Installation System Administration

of Windows NT Conference

July 14–17, 1999
Seattle, Washington, USA

Conference Organizers

Conference Co-Chairs

Gerald Carter, *Auburn University*
Ralph Loura, *Cisco Systems, Inc.*

Program Committee

Ian Alderman, *Cornell University*
Jeremy Allison, *Silicon Graphics, Inc.*
Paul Anderson, *University of Edinburgh*
Phil Cox, *Computer Incident Advisory Capability*
Alan Epps, *Tektronix, Inc.*
Aeleen Frisch, *Exponential Consulting*
John Holmwood, *TransCanada Pipeline Ltd.*
Matthew Olguin, *Complete Data Solutions, Inc.*
Andrew Rieger, *Lehman Brothers, Inc.*
Martin Sjölin, *Warburg Dillon Read*

The USENIX Association Staff

Contents

The 2nd Large Installation System Administration of Windows NT Conference

July 14–17, 1999 Seattle, Washington, USA

Message from the Program Co-Chairs

Welcome to Seattle and the 2nd Large Installation System Administration of Windows NT conference!

When we first issued the call for papers for this year's conference we posed the question, "What are the qualities of good models of system and network administration?" The belief was that there are certain elements of administration that are independent of the environment in which they are employed. We feel that the papers published here, the work of twenty-six authors, represent an excellent response to that question. These papers are the sharing of solutions to problems we all face: "How do I roll out a given application to all my client desktops?" or "What are some methods I can use to secure this NT server in my environment?" We hope that as you read these Proceedings, you will find ideas for new solutions and will share those with us next year.

We would like to personally thank all the members of our program committee. Each person was an invaluable help in soliciting paper submissions, contacting speakers for invited talks, and providing input for the direction of this year's LISA-NT.

As a final note, we would like to recognize Ellie Young, Jane-Ellen Long, and the entire USENIX staff. It is very easy to take for granted all the work that they do in organizing a conference such as this one. Their guidance, advice and, sometimes, constant prodding were always appreciated.

Cheers,

Gerald Carter and Ralph Loura

Conference Co-Chairs

Scalable, Remote Administration of Windows NT

Michail Gomberg, Craig Stacey & Janet Sayre
Mathematics and Computer Science Division, Argonne National Laboratory

Abstract

In the UNIX community there is an overwhelming perception that NT is impossible to manage remotely and that NT administration doesn't scale. This was essentially true with earlier versions of the operating system. Even today, out of the box, NT is difficult to manage remotely. Many tools, however, now make remote management of NT not only possible, but under some circumstances very easy. In this paper we discuss how we at Argonne's Mathematics and Computer Science Division manage all our NT machines remotely from a single console, with minimum locally installed software overhead.

We also present NetReg, which is a locally developed tool for scalable registry management. NetReg allows us to apply a registry change to a specified set of machines. It is a command line utility that can be run in either interactive or batch mode and is written in Perl for Win32, taking heavy advantage of the Win32::TieRegistry module.

1. Overview of MCS Environment

The computing environment in the Mathematics and Computer Science Division of Argonne National Laboratory consists of nearly a thousand computers, including supercomputers, servers, workstations, desktop machines and laptops. Our NT infrastructure consists of 10 production NT servers, 8 Samba servers, 8 experimental NT cluster servers, approximately 70 NT workstations, and approximately 30 machines running Win95/98, most of which are laptops.

The systems group that manages these machines consists of 8 full time administrators, 3 of whom are at least partially responsible for the Windows environment. As with all other systems administrators, we have plenty to do and our users' expectations are high. Our environment continues to grow in complexity and scale; thus we are constantly searching for better and more scalable techniques for managing our collection of hardware and software. To that end, we have installed commercial packages such as HP OpenView, and have resorted to writing our own software when other packages did not fit our working model. Fortunately for us, the techniques for scalable management of machines running variants of UNIX are fairly well understood. We spent a considerable amount of time trying to find similar techniques that could be applied in a Windows NT environment. We did this partly as an academic exercise, but primarily because our NT environment is large enough that we could save significant time and effort if many mundane management tasks were handled remotely.

2. UNIX vs. NT Remote Management

UNIX machines come with many tools that make remote management possible. Many of the tools such as telnet/rsh daemons and rdist are provided by the vendor and are considered to be part of minimum standard installation. Remote management for UNIX machines, in many cases, involves obtaining a shell via rsh or telnet, editing configuration files that control the operation of the machine, and possibly restarting daemons. In some cases low level console support exists, allowing one to manage aspects of the hardware, such as rebooting, via a serial line. These methods alone allow us to manage all of our UNIX servers from a single console. We can connect to any machine, work with a familiar environment, and change any aspect of the machine configuration as long as we know what file to edit.

As our NT environment grew, we became very concerned about remote access issues. We could imagine running an environment in which we would have to walk to the machine room simply to add a user, or run from server to server trying to figure out why some service is failing. We also were concerned about the lack of command line administration tools. Our ability to manage a large number of machines effectively depends on the ability to automate tasks. We usually do

this by scripting complex or repetitive operations, but we didn't see a way to accomplish this under NT.

As our experience grew, we realized that NT actually had many of the remote access primitives built in, but the model for remote access on NT was markedly different from UNIX. Instead of requiring a remote shell on the machine, we could manage aspects of the network using C/C++ code. A simple example of this is the NetUserAdd() system call, which creates a new user either on a target machine or in a domain. We discovered that NT has many of such system calls for managing the network, accessing the registry remotely, shutting down remote systems, etc. These calls generally either connect to a remote machine first, or name the target host in the first parameter of the call. The calls either perform the action or return some opaque handle. Additional calls then can be made to perform more operations using the handle as a parameter. The actual underlying mechanisms for the connection tend to vary. Some are implemented via RPC, others via NetBIOS functions. However, in most cases the underlying mechanism is never revealed in the documentation.

These functions provided us with the ability to do some remote management, but this method doesn't scale well. We didn't have the time to write management suites in C, and those of us who are familiar with C didn't have the necessary experience writing Windows code. Finally, even if we did invest time writing this code, it would not have helped us on the UNIX side. Our UNIX management code is written almost exclusively in Perl and uses a different set of primitives for remote communication. Fortunately, Microsoft was building some of this functionality into many of its own management tools. However, the ability to manage machines remotely was neither well-documented nor emphasized, probably because Microsoft was not considering enterprise issues at the time.

3. Remote Management Issues

We must consider what types of things we need to manage. The following is a non-exhaustive list of the major issues.

- User account/security management. This task consists of adding and deleting users and groups from the global or local user list, and changing user access privileges to specific resources, such as files or machines.

- Service configuration and management. Many of the services that we run need to be configured for our specific environment. Because of open standards, many of the services function in similar ways on NT and UNIX. For example, web daemons, DHCP servers, and DNS servers all need to be configured continually in a growing environment.

- Exception notification and logging/security audit. Both hardware and software generate exceptions and notification messages that warn of failure or other problems. Messages that show user access violations (repeated failed logins) and messages that warn of impending hardware failure (repeated failed writes to disk) are of particular interest.

- TCP/IP stack and client configuration management. A task we recently faced was changing the netmasks on most of our machines as we flattened our routed network to a switched environment. Although these tasks are not performed often, they usually have to be performed on many hosts at the same time.

- Shutdown/reboot. There are many situations in which we need to reboot machines remotely. This is especially true with NT. The netmask example above required a reboot for the change to take effect. We also have needed to shutdown all machines for planned power outages.

- Software configuration. Some software packages are run by a specific group of users. For example, our administrative group runs a custom application for ordering equipment and supplies. These packages sometimes need global configuration changes, such as moving a database server to a different machine.

- Software distribution. Software distribution is mostly concerned with keeping the software versions on each individual host up to date. A common misconception has this problem limited to Windows NT because software is often shared in a UNIX environment. In reality, the problem is simply a larger one on NT because almost all software resides locally. UNIX also has local software, but it is often limited to vendor patches or small confined packages that have limited scope such as a local version of a shell executable.

- Troubleshooting/debugging. When software or hardware fails, the cause of failure is not always immediately evident. On the UNIX side, we often log on, look at running processes, kill resource

hogs, or, in the case of hardware failure, we often can reconfigure the OS to provide a reduced level of service so that the effect of the outage is minimized. NT generally requires us to perform similar tasks, although we often find that our software tool set, while functional, is more limited in coverage.

- Automated tasks. Automated tasks perform required daily maintenance or periodic routine jobs. An example we employ on our NT servers is a script, run nightly, to copy IIS log files to our UNIX servers where another script is run to calculate usage statistics.

4. Scalable Remote Management

The problem with remote management on UNIX or NT is that, in its simplest form, it's not really scalable. Using telnet or rsh alone is no better than sitting at the console of the machine. The ability to manage all such machines from a single console, one machine at a time, is no more convenient. Tools that provide good, simple abstractions for the environment and allow entire groups of machines or services to be managed as one allow us to manage that environment in a scalable way. A simple example using such tools is a file containing a list of all the hosts, together with a script that reads the file, issuing a remote command (rsh) to each host. In this example, the file represents the entire environment, and rsh provides a simple remote management protocol. Another example of a simple but widely used tool in UNIX is rdist. Rdist allows a single file, such as /etc/passwd or the configuration file for TCP wrappers, to be distributed to multiple hosts.

Some management scalability on UNIX machines is achieved via file sharing through NFS. The simplest example of this is an NFS mounted /usr/local, which allows software to be installed and configured only once for all the client machines. A more complicated example in our environment is our global Samba configuration file, which is merged with local configuration files on the servers, allowing us to manage our Samba servers as a group.

The aforementioned management techniques suffer from another problem. They are not really aware of the environment as a whole. All require explicit knowledge of which files need to be edited, and may require multiple versions of a single file to be distributed to different classes of machines. Thus, they don't significantly shield the administrator from the complexity of the environment.

Slightly better approaches are used in our "Config" system, written by Remy Evard, and in Mark Burgess's cfEngine. [1][2] Both of these tools are still somewhat file- centric (our Config system currently defines over 100 files). The strength of these tools is that they allow system management based on abstract classes such as OS type, machine function, or other arbitrary classifications. These tools describe the system as a whole and manage machines by making them comply to the desired standard. CfEngine is also different in its lack of dependency on built-in UNIX communication primitives. While the Config system relies on rsh and NFS, cfEngine uses its own daemons and communications protocols.

The scalability of our UNIX management approach is achieved through the availability of a highly diverse tool set. We have many traditional tools that allow us to manage single machines and a new generation of tools to manage the whole environment. The common themes for these tools are a simple, built-in communication method and a building-block approach to create more complex behaviors.

5. Tools for Remote NT Management

Today we can manage most of our NT infrastructure from a single console. We still make infrequent trips to the machine room, but most of those are necessary because of hardware issues and not day to day management. The most important tools at our disposal are the Ataman telnet/rsh daemon, ActiveState's port of Perl, and the NT Resource Kit. These tools allow us to manage accounts and services, make registry changes, distribute software, and reboot machines. Other tools included with NT and the NT Resource Kit allow us to view event logs, monitor process state and resource utilization, capture and filter network traffic, and remotely connect to the console when required.

The Ataman telnet service and Perl are a huge advance for the NT world because they provide a familiar management environment. We recently used the following script to shutdown all of our NT workstations in preparation for a planned power outage. Interestingly, this script was triggered using rsh from a UNIX workstation. Also note that portions of the NT Resource Kit are copied into a DFS share so they are accessible to all of our NT machines via a UNC path. This is a similar concept to /usr/local in our UNIX environment.

```
use Win32::NetAdmin;

Win32::NetAdmin::GetServers(undef, "MCS", SV_TYPE_NT, \%all_server_ref);

Win32::NetAdmin::GetServers(undef, "MCS",
     SV_TYPE_SERVER_NT | SV_TYPE_DOMAIN_CTRL | SV_TYPE_DOMAIN_BAKCTRL,
     \%nt_server_ref);

# get rid of servers and only leave workstations
foreach $server (keys(%nt_server_ref)){
    print "$server\n";
    delete $all_server_ref{$server};
}

foreach $server (keys(%all_server_ref)){
    $result = `\\\\mcsnt\\DFS\\soft\\adm\\packages\\ntreskit` .
            `\\shutdown \\\\$_ /T:60 /Y /C`;
    print $result;
}
```

Another common task is changing a registry value on all machines. The following example copies AT&T Research UK's VNC settings (including the password) from the host machine to all other machines on the network. This script can be run on any of our machines.

```
use Sys::Hostname;
use Win32::TieRegistry;
use Win32::NetAdmin;
$Registry->Delimiter("/");

Win32::NetAdmin::GetServers(undef, "MCS", SV_TYPE_NT, \%all_server_ref);

my($hostname) = uc(hostname());
$hostname =~ /^(\w+)\./;
$hostname = $1;

my($src_key) = $Registry->Connect($hostname,
     "Users/.DEFAULT/Software/ORL/WinVNC3");

foreach $server (keys(%all_server_ref)){

     next if( $server eq $hostname );
     my($rem_dst_key) = $Registry->Connect($server,
          "Users/.DEFAULT/Software/ORL/WinVNC3");

     if(!$rem_dst_key){
          $rem_dst_key = $Registry->Connect($server,
                                  "Users/.DEFAULT/Software/ORL");
          next if(!$rem_dst_key); # host doesn't have VNC at all
          $rem_dst_key->CreateKey("WinVNC3");
          $rem_dst_key = $Registry->Connect($server,
                              "Users/.DEFAULT/Software/ORL/WinVNC3");
     }

     my(@value_names) = $src_key->ValueNames;

     my($value, $value_string, $value_type);
     foreach $value (@value_names){
          ($value_string, $value_type) = $src_key->GetValue($value);
          $rem_dst_key->SetValue($value, $value_string, $value_type);
     }
```

```
        undef $rem_dst_key;
}
```

Conceivably, both of those tasks can be accomplished on our network by hand. However, very large or distributed organizations could certainly benefit from this more scalable approach. The above script 65 seconds to complete on 126 hosts on our network, which includes bandwidth ranging from switched 100 Base-T to 128kb/s ISDN. Expanding this to a network of 5,000 hosts, the script would take approximately 43 minutes to run, significantly less time than it would take to visit every machine in person.

While the UNIX management model is file-centric, the NT management model is almost entirely registry-centric. Although the registry is a binary file, logically it is arranged as a hierarchical forest much like a POSIX directory structure. This similarity allows some of the techniques that we learned from our UNIX experience to be applied to NT management. For example, we can extend the Config system or cfEngine (if ported) to manage a set of registry keys. Unfortunately, Microsoft has not completely committed to using the registry as a central and sole store of configuration information. A notable exception is IIS, which uses its own binary format file as well as the registry for configuration data. One positive direction that Microsoft seems to be taking is the ADSI style management interface. This technology relies on OLE objects addressed in the form of PROVIDER://<Host>/<object1>/<object2>/<etc...>. Each returned object manages a particular aspect of a service or a machine. Because the objects are OLE-based, they are language neutral and can easily be used from Perl. Since the objects can specify remote hosts, they can be used as building blocks for higher level scalable tools.

There are, however, several problems with this approach. Since ADSI objects rely on OLE, the objects must be known in advance by the client workstation that is doing the management. This means that for true scalability, all the known objects have to be distributed to all the machines. Since these objects are usually a part of a much larger package, such distribution is impractical and may break license agreements. Another problem is that not all management functions are covered, so changes may still have to be made through the registry or through configuration files. The final, and possibly most difficult, problem is that because ADSI is not available on UNIX platforms, we can't use our NT developed tools for integrated enterprise management. We hope to avoid much of the duplication of effort that goes into managing these two environments.

Many of the other tools we use for remote management come with NT. For example, we use the User Manager and Server Manager tools to manage accounts, services, and shares. We use Regedt32 for remote, single-machine registry edits. We often use the Event Viewer as a first step in our trouble shooting, and we use the Performance Monitor as both a data collection tool and a first line of defense against problems. If necessary, we use VNC from AT&T Research UK to connect to the console remotely and perform operations that can only be done from the console.

6. The NetReg Tool

The motivation for NetReg came from writing many Perl scripts to manage different portions of the registry. We realized that we needed a tool that was easily scriptable by someone familiar with the registry but not necessarily with Perl. We wanted a tool that could be used from the command line (in a telnet window), did not depend on regedit, and could be used in interactive mode to make individual changes. We first looked at the NT Resource Kit, which contains at least 14 different tools to edit the registry. Each tool uses a slightly different syntax, and all work in slightly different ways. Only some can be used to manage remote registries, and only a few support scripting. We wanted to have a single tool that combined all these functions, had a simple unified syntax, and could build on the power of Perl.

NetReg is built around the Perl TieRegistry module written by Tye McQueen. The TieRegistry module is extremely powerful and easy to use. It lets you manipulate the registry via an object or via tied hashes. The "chaining" feature allows intermediate registry objects to be created, which then can then be used for subsequent connections to sub-objects. TieRegistry also allows remote connections, can manipulate any type of registry value, and can cache lookups for enhanced performance.

In its current incarnation, NetReg is in many ways a proof of concept. Both the syntax and the execution model will change in further releases. The current syntax consists of the following elements.

```
load [net <domain>|sms|file|machine_list] -pick -regexp
search [keys|values|valuename|all] -r -pick -regexp <path> <search value>
copy [keys|values|valuename|all] [<identifier>|<src_path>] <dst_path>
list [keys|values|valuenames|all] [<path>|<identifier>]
select -regexp -pick <identifier>
edit <identifier> <perl_regexp_replacement>
reboot <identifier>
bookmarks [load|save|list] -pick -regexp
bm <bookmark_name> <path>
```

As stated before, NetReg can be run in interactive mode or in batch mode. The interactive mode is currently command line driven but will be extended in future versions to support a TCL/TK GUI as well as a command line.

Most NetReg commands return an implicit result. The succeeding operations use and augment the result. For example, a load operation creates a list of machines; a search operation uses the current machine list and will return a list of machines and keys that match the search criteria; an edit operation simply uses the current machine and key list to perform a global edit operation. The following example illustrates the usage. This three-line sample performs the same VNC registry copy as the previous Perl script.

```
$vnc = LMachine/Users/.Default/Software/ORL/WinVNC3/*
$all = load net MCS
copy values $vnc $all/$vnc
```

A very useful feature of NetReg is the ability to use bookmarks. Bookmarks can either be a simple alias for a complex key name or can consist of a complicated chain of registry keys and values. The following examples demonstrate this.

```
bm vnc LMachine/Users/.Default/Software/ORL/WinVNC3
bm services "LMachine/System/CurrentControlSet/Services"
bm netcards <services>/NetBT/Adapters/*
bm ipaddr  <services>/{<netcards> =~ /\/(\w)$/}/Parameters/Tcpip/IPAddress
```

In the example above, *vnc* and *services* are simply aliases for longer key paths. The bookmark *netcards* is an alias that may return multiple paths or values. The *ipaddr* bookmark is special. The curly brackets denote an executable query. Thus, *ipaddr* will first evaluate the *netcards* query, then apply a Perl regular expression to the result. For Perl novices, the regular expression selects the last word after the "/", which in this case should be the name of the device driver which implements the NetBT service and, in most cases, the TCP/IP service as well. The result of the regular expression match then is inserted into the bookmark. Finally, the bookmark is evaluated as a whole to return the IP address assigned to the adapter. In machines with multiple adapters configured with TCP/IP, the same query would return all the IP addresses assigned to each card.

The bookmarks can be used in any command that accepts a path. To delimit the bookmarks from the rest of the path, they are enclosed by "<>". The bookmarks' usefulness depends on the user. While there will be some predefined common bookmarks in the distribution, a judicious use of additional bookmarks will help simplify registry navigation.

Since NetReg is still evolving, it is premature to discuss all its features in this paper. Additional documentation and examples will be available on-line in the distribution.

7. Other NT Remote Management Solutions

We took an informal look at other Argonne divisions and one educational site to discover how other groups were coping with the problems that we encountered.

Some sites find they are able to use NT efficiently without adding any outside tools. For example, Argonne's Office of the Chief Financial Officer has about 20 servers (for development, files, and applications) and approximately 300 workstations, all managed by the tools that come with NT. They have to visit the machines in person occasionally, but find that most remote management tasks they need to perform are available. They evaluated SMS, and plan to

start using it when the new version becomes available, but have managed without adding to NT for quite a while. [7]

A little further along the continuum lies Concordia University. While they use no remote management tools for NT servers located on their main campus, they have added to NT's remote capability by using PCAnywhere and CarbonCopy to install and monitor NT at other locations. Unfortunately, a phone call to the person at the console is still the solution to some problems. For managing workstations, KiXtart is used for small changes. Major changes are made by cloning machines to identical states using GHOST. This works in that environment since these are lab PCs. Concordia does have concerns for the future, however, and hopes to solve the remote administration challenge more satisfactorily, since they are due to become the central support for a consortium of colleges spread across the country. [8]

Argonne's Electronics and Computing Technologies (ECT) division goes a little further still. They used SMS in the past, and plan to use it again when version 2 is released with the new version of BackOffice. They had a successful installation of Office 97 using SMS to push the install, but currently they must physically visit a machine to install software. They remotely manage servers by mounting administrative shares from the server to the workstation. Most tools (user manager, server manager, etc.) are designed to run remotely, and the administrators find they can do everything they need. They're reasonably happy with their existing solution, but have encountered some tasks that simply do not have an obvious remote solution, *e.g.*, installing FrontPage extensions remotely. ECT manages three main domains, with about 24 servers. Users have administrator privileges, even though SMS is used for installs, since installs run as the user.

One ECT project did turn out to be a solution to the remote install problem. Autolink. an Argonne developed tool, was designed to solve the problem of different programs requiring different versions of a .dll file, but it became a solution to the remote install problem. Managing the .dll's by keeping them on the server meant that a solution had to be found for installation on the remote clients. Although it is narrow in scope–only designed to handle a well-defined set of applications and requires user input to execute–Autolink's combination of Oracle and PowerBuilder allows entire groups of applications to be managed from a single location. [9]

An even heavier user of SMS is Argonne's Chemical Technology division. The systems administrators customize SMS heavily, writing scripts that run on the desktop to allow remote installation of anything they want. This is an extremely centralized environment, with workstation users denied administrative privileges on their machines. Despite the authoring efforts put into remote installation on workstations, server management in that division is accomplished by the tools that come with SQL manager, user manager, DCHP manager, etc. [10]

8. Unsolved Remote Management Tasks

We have encountered some management tasks that we simply cannot perform remotely because of current NT and hardware limitations. Some issues we simply decided to ignore, partially because our environment permitted it, and because the payoff for solving those issues at our site was small.

8.1 No locked down machines

We run our machines using a trusted environment model. We operate under the assumption that if you have physical access to the machine, you have the ability to get root-level access on that machine. Because of this, the local administrator account gets no domain privileges. In fact, whenever a user is assigned to a particular workstation, we grant administrator access right away. We experimented with not doing this, but it proved to be more trouble for us as well as the users because they couldn't get their machines to do what they needed to when they needed it.

We have found that allowing users to have administrator access has not been as problematic as we originally assumed. The users are warned not to keep any data on the machine's disk, and to expect that a full rebuild is a possibility if they mess up the machine to the point where we can't fix it. This has rarely been the case, but it has happened.

8.2 User-Installed Software

In giving the users administrator access to their machines, the distribution of software has become even easier for us. For most packages, we have set up a "come 'n' get it" system, where all of the software we distribute to the users is available from a central place—— in our case, a web page.

We created a Windows Software Repository web page (http://www.mcs.anl.gov/windows). It's a

manually updated web page that uses UNC hyperlinks to direct users to the setup programs for various packages. The packages are distributed from a DFS shared repository and the web page includes any helpful hints for installing the software, as well as license keys. We've also taken advantage of the automated installs we've created for some packages for use in the machine building scripts. We provide links to those for users who wish to have a default install on their Windows NT machine.

In order to allow users to take full advantage of this web page, independent of their browser or OS (Windows 9x on laptops vs. Windows NT on desktops), each network-resident distribution directory has a setup.bat, which calls the appropriate setup program. This ensures the setup program is launched with the correct path, as Netscape does not pass the path along when directly running a "file:" hyperlink. Using this setup.bat method also allows us a crude but easy installation tracking method, by sending an email using the freeware email sending program BLAT(http://www.interlog.com/~tcharron/blat.html) whenever the installation program is run. This is especially helpful in dealing with software that is not under a site license. Because we use UNC links to a DFS shared directory, we maintain security in that only users logged into the domain can access the software directory tree.

9. Remaining Problems

While programs like SMS remote control and AT&T Research's Virtual Network Computing help us get to the console of machines that are up, we still do not have quite the low-level console capability we have in UNIX. If our servers get stuck in the boot process before Windows actually starts, we are left in that awful position of physically having to be at the machine, which is not very practical on a weekend. While this will still happen with our UNIX machines from time to time, it is still far less frequent an event than in with PCs. Still, it makes a remote reboot a little more risky.

Our current procedure when an NT box can't boot is to boot it with ERD disks at the console. It would be incredibly useful if Microsoft implemented ERD-like functionality into NT and allowed the console to be transferred to a serial port.

10. Summary

Our experience with NT has shown us that, with the right mix of tools, it is possible to manage NT remotely, in some cases in a scalable way. NT has a different set of primitives than UNIX for remote communication. It looks like Microsoft is taking enterprise management issues much more seriously. We hope they will learn from the UNIX community how to handle large installations of machines and software.

We also hope that Microsoft will take more of a building block approach to providing management applications for NT. Experience has shown repeatedly that monolithic applications have too great a learning curve and are often too inflexible. We also would like to see Microsoft stick with a particular set of its own standards, such as using the registry or ADSI, so that we can more easily build tools to our own specifications.

11. Author and Project Information

Michail Gomberg is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. He was the lead technical architect for this project. His email address is gomberg@mcs.anl.gov.

Craig Stacey is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. His email address is stace@mcs.anl.gov.

Janet Sayre is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. Her email address is sayre@mcs.anl.gov.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

12. References

[1] R. Evard. An analysis of unix system configuration. Proceedings of the 11th Systems Administration conference (LISA), page 179, 1997

[2] Mark Burgess, Cfengine A Site Configuration Engine, USENIX Computing systems, Vol8, No. 3 1995

[3] Microsoft Windows NT Workstation 4.0 Resource Kit, Microsoft Corporation, Microsoft Press, 1996

[4] Comparing Microsoft Windows NT and UNIX Remote Management. Microsoft White Paper.

[5] Comparing Microsoft Windows NT and UNIX System Management. Microsoft White Paper.

[6] Microsoft Windows NT from a UNIX Point of View. Microsoft White Paper.

[7] Kay Burdi, Argonne National Laboratory-OCF, 5/26/99

[8] Rich Helmke, Concordia University, 5/26/99

[9] Rich Raffenetti, Argonne National Laboratory-ECT, 5/28/99

[10] Steve Gabelnick, Argonne National Laboratory-CMT, 5/27/99

A Networked Machine Management System

Dave Roth
rothd@roth.net
Roth Consulting
http://www.roth.net

1 Purpose

Out of the box, Win32 machines lack the ability to be efficiently managed. It is up to an administrator to implement some form of system management. For many administrators with relatively small networks this means walking from machine to machine to install new software or check to see how many CDROM drives exist on the computer. Considering the sheer size of some networks it becomes a nightmare to do something such as deploying an updated driver or service pack. Even a simple task as discovering which computers have enough free hard drive space for a software upgrade can take many hours of an administration teams time.

To be sure, this is not an isolated problem. It is ubiquitous enough to create a market where there are several commercial products available to help administrators manage this issue. Microsoft's Systems Management Service (SMS) is one example. However a product such as this can become quite time consuming just to maintain the system as well as costly for full licensing.

With a little bit of forethought administrators can implement a simple and efficient system for performing the same functionality as a full-blown SMS implementation. By using off-the-shelf products and some of Win32's standard services a network of almost any size can be managed.

This paper provides a case study on one particular implementation.

2 Needs Assessment

As with any project a needs assessment is a wise beginning. This paper focuses on a large cellular service provider which we had the pleasure of consulting with. We shall call this company *CellBell*.

2.1 Background

CellBell had a network of approximately 1700 NT boxes scattered across the state. The machines were running a combination of NT 3.51 and NT 4.0. Some had different service packs installed while others had none. Some machines had 32 megs of ram while others had only 16. Some had large hard drives with plenty of free space while others had practically no space available. Some had 486 processors while others were Pentiums. The point is that the network was non-homogenous and there was no documentation to help sort out how each machine was configured. To make matters worse the IT team was competent but under funded. There was no money to send the team to SMS training let alone money to purchase SMS licenses.

From time to time CellBell corporate headquarters would decide that all machines on the network would have to have a software update. The team, loyal to a fault, would painstakingly walk from machine to machine, log off any current user, log on as administrator, check if the machine satisfied any ram and drive memory requirements, connect to a network server, install the software then reboot. If the capacity of the machine did not meet the required spec then it would be added to a list then later parts would be procured and installed. This could take days to accomplish – requiring overtime and travel.

2.2 Needs

We decided that there was a need for a system which would:

- Gather information about each machine in a timely basis
- Effectively install software on a machine with minimal human interaction
- Perform routine machine maintenance (update drivers, remove temporary files, etc)
- Alert the IT team when a machine was having problems
- Be inexpensive
- Be flexible enough to adopt to any future need
- Be easy to use

Considering these basic issues and a rather tight budget the team set out to develop an infrastructure that could effectively provide all these needs.

3 The Infrastructure

After much deliberation an infrastructure was devised. To be sure it is far from a perfect design but for a tight financial and time budget it worked well.

3.1 Design Overview

The entire process is quite simple. It makes use of what we called *scripts*. A script could be anything executable such as a program or a batch file, for example.

At some predetermined time each machine on the network would spawn a process which and check to see if there were any scripts in a predetermined location (presumably on a network server). Each of the scripts would then be executed. Once all scripts have been run the process ends.

That is it; quite simple indeed. The system would be only as complex as the administration staff decided to make it.

3.2 Scripts

Each script can do anything that the administrator may need. One may look for a given ODBC driver on the machine and install it if need be. Another script may install some new software package. Another script may examine the local event log for errors and warnings. Yet another script may download and implement an anti-virus programs updated virus signature files.

So this simple system could have scripts written which modify registry settings, copy directories, apply permissions to files, synchronize the clock with a server or anything else one may need.

3.3 Script development

The management system was going to be home grown to avoid the costs associated with commercial software. Since nobody wanted to even consider using DOS like batch files a programming language would have to be chosen. Due to it's remarkable abilities and reasonable price (free) the Perl language was elected.

Perl is well known as the Swiss army knife of languages. It is almost as powerful as C++ but has a much faster development cycle. And if the language has any limitations they can be eliminated with Perl extensions (plug-ins developed in another language such as C).

One other compelling reason to use Perl is the vast number of modules and extensions that have already been created for it. If there is a need to access web servers, ftp files, search for files and directories or even query databases you will most likely find that someone has already written an extension to do just that. Of which you can download from the net.

It was decided that the system engine and its subsequent scripts would be written in Perl.

3.3.1 Script Execution Tracking

Undoubtedly some scripts may be designed to only run once. If, for example, a software package is to be installed the install script needs to only run once. It would be simple enough to create the script and let every machine run it then remove the script. However, if a new machine is brought online it, too, would need to have the software package installed.

If each install script added an entry to some configuration file on the local machine the script could check to see if it had already been run. This execution tracking permits scripts to remain active in the repository but not a menace to machines already having been affected by it.

3.4 The Script Repository

The infrastructure was designed around a central script repository. This is simply a network share and directory where all the scripts are located. By locating the scripts in one central location the burden up altering scripts is greatly reduced – there is no need to alter a script on every computer or on several servers. This directory can be part of a replication tree so that any change made will be replicated to other backup repository machines.

CellBell had a WAN that extended across the state with multiple subnets. Since many of the subnets were connected together using a slow frame relay link it was crucial that the network design include a repository on each subnet. Ideally there would be one repository on each backup domain controller this way the name of the machine can be discovered by locating a domain controller.

The repository was to be the hub of the management system. The bulk of the files accessed were to be through this repository. The good news was that since the access consisted of simple reading of files there would be very little overhead for the repository machine. Since the BDC's were configured to only act as a domain controller (for user authentication) the overhead of serving such files made only a minimal impact.

By utilizing domain controllers CellBell also provided redundancy. If the repository goes offline then the

system engine would locate another domain controller, which may be on different subnet.

3.5 Database Services

One crucial component of this system was a database server. Our team of administrators needed a reliable way of generating an accurate and up-to-date list of attributes regarding the machines. Attributes such as hard drive sizes (physical and available), amounts of ram, pagefile size, processor speed and service pack level just to name a few. A database server was installed to maintain this information.

Since the database was populated with accurate information we could submit queries to discover which machine had less than 64 megs of ram, for example. This provided an effective way to access a wealth of administrative information from across the network. There was no longer a need for a remote administrator to call into the help desk to obtain such information; she could just simply submit a query to the database server. This proved to be absolutely invaluable when management needed fast answers regarding our current state of affairs.

3.6 Information Server

Since we had an accurate database of information the next step was to install an intranet web server. We had written CGI scripts that provided access into the database. It was this way that we empowered non-technically inclined folks (management in particular) to query information regarding the network.

This tool immediately became indispensable for both management and the administration team. Administrators were able to logon when they came in and view reports of the network that were generated earlier in the morning. If an administrator was not able to make it into the office she could dial in remotely and peruse the logs.

4 The Details of the Design

The actual engine and scripts would be fairly quick to prototype but it is the other non programming aspects that would prove to be more difficult. The details such as how to schedule the engine to be run on a timely basis and how to secure the system were the issues that took the most time to work out.

4.1 The Scheduler Service

The Scheduler service works very much as its UNIX cousin the CRON daemon. Basically speaking the Scheduler service allows a job to be scheduled for a given time. A job is simply a command that can be spawned as a process.

The management system will make copious use of the Scheduler service. A job is added to the service that will execute the main Perl script (the system engine). The job is scheduled for a time in which it will not interfere with a user since occasionally a script may need the computer to reboot.

At CellBell we configured each machine to run the engine between 3:00 and 4:00 in the morning. When the job was scheduled a random time was chosen so that there would be less of a burden to both the network and the repository as all the machines woke up.

4.2 Security Concerns

One of the primary concerns regarding the implementation was that of security. After all it is quite easy to write a script that deletes all files on a hard drive, and if some disgruntled employee slipped such a script into the repository there would be quite a problem. Securing this system was absolutely imperative.

4.2.1 Scheduler Account

By creating a special user account that the Scheduler service would run under we were able to guarantee security. In our case we created a user account called "SchedulerAccount". The decision for this name was quite arbitrary but self-descriptive.

The "SchedulerAccount" account was made a member of the "Domain Admins" group. Since each machine in the domain included the "Domain Admins" global group in its local "Administrators" group this gave the Scheduler service administrative abilities on every machine.

4.2.2 Repository Permissions

The script repository was limited so that only "Domain Admins" and "Administrators" had access to modify the files. To protect entry into the repository the permissions were applied to the sharepoint. To also protect against backdoor access the files and directories in the repository tree were also protected with the same permissions as the share.

There was a debate as to whether or not permissions for users should be applied. Some administrators felt read only administrators should have access whereas others did not foresee a problem with users having read only permissions. A decision to prevent users any access

was made so that a script could have hard coded userids and passwords. Whereas any file containing userids and passwords is a potential security risk it was decided that in some instances it was necessary.

Considering that users are usually more curious than they need to be a decision was made to hide the share. If a user does not see a share there will be no desire to play with it. The share was hidden by appending the share name with a dollar sign such as "Repository$".

4.3 Variations Of Scripts

A multitude of Perl scripts were written which performed an amazing array of functions. Some of the more useful ones included:

- A script which compared the machines copy of Perl against the repositories copy. Any updates were then replicated down to the local machine.
- A script checking for files in the temp directory. If files exist that were older than 2 weeks they were removed.
- A script that retrieved all IP addresses hard coded on the machine and submitted them to the database. At the time CellBell was using hard coded IP addresses, none of which were mapped to physical machines. This script helped the IT team track IP address with userids and machine names.
- On script modified each local machines registry to change a hard coded IP address to use DHCP.

There were three scripts in particular, though, that proved to be most useful. These performed ODBC client database configuration (setting up the data source name so that the scripts could talk to a database), collecting machine information and checking the event logs.

4.3.1 ODBC Databases

Since our implementation of the system made use of a SQL Server database it was necessary that there was a verification that the SQL Server ODBC driver was installed and that the correct data source name (DSN) was configured.

The file _ODBC.PL (*Script 3*) performed this task. Notice that the file name begins with an underscore. This was to guarantee that it was run before any other script. Since many of the scripts would attempt to connect to the database it was important that this script was the first (or at least one of the first) scripts to be executed.

4.3.2 System Info

One of the most important scripts that were created was the OS.PL script (*Script 4*). This script collects information regarding the physical machine and the operating system. This information is submitted to the database for later processing.

The script collected information about the computer such as:

- Lists of all local hard drives, CDROMs and removable drives, their total drive size and how much space is available.
- The amount of physical RAM.
- The total pagefile size.
- The OS type (Win NT or Win 95/98).
- The OS version.
- The OS service pack.
- The class and speed of the processor.

Once this information had been collected it was submitted to the database where the information could later be processed.

4.3.3 Event Log Scanning

With thousands of machines on the network it became virtually impossible to keep an eye on each computer. With the EVENTS.PL script (*Script 5*) the local event log was scanned for warnings and errors. These events were then submitted to the database. In this way the administration team could generate reports consolidating multiple errors and alert the staff which of the machines were having problems.

4.4 Installation

Installing the system on each computer promised to take time. In the case of CellBell the machines on the network were not setup with much forethought.

One of the first things we needed to do on each machine was to setup the Scheduler service to logon using the domain account "SchedulerAccount". Additionally we needed to copy the Perl tree onto the local machine then secure it such that users only had read permissions. This required that the drive be formatted as NTFS. If it was a FAT drive the installation routine would need to run the NTFS conversion utility.

A plan was devised in which our team would walk and touch each machine, logging on with a temporary administrative account we created for this sole purpose called "InstallAdmin". This account was added to the "Domain Admins" global group. Since each machine included the "Domain Admins" group in its local

"Administrators" group the logon would have administrative authority over the machine.

The "InstallAdmin" account was configured with a logon script which acted as the install script (refer to *Script 3*). The actual logon script command was set to:

"\\server\perlshare$\perl\bin\perl.exe \\server\perlshare$\install.pl"

This would run both Perl and the install script from the repository.

Since the script would configure the machine then reboot all that the administrator had to do was logon with the "InstallAdmin" account then walk to the next machine.

The installation script would register the Scheduler service to logon using the userid of "SchedulerAccount" and the provided password. Since the password would be hard coded in the Perl script this particular file was set with read permissions for only the Administrators and no access for everyone else.

Since all of the functions that the installation script performed could be accomplished remotely it is possible to have a machine or machines on the net walking through the list of online computers and remotely configure them. In the case of CellBell none of the machines had NTFS partitions and since the convert.exe utility must run as a process local to the machine we decided to physically log onto each computer. This way we could also boot any computer that had been powered down.

5 Conclusion

Systems management programs can make life easier for any administrator who has a large network of Windows NT machines. But since the cost and learning time is not necessarily something that is affordable for everyone it makes sense to get the most out of what is available.

The infrastructure presented here not only works but it also works well. The ability to update files, clean directories, apply permissions and other such things is enough to justify the effort.

For CellBell, the total time to create the infrastructure took one person roughly 2 days. Once the installation account was created and configured the system was up and running on all machines in just a couple of days.

Within the first month of use the team was given the task to rollout quite a large and new application. The amount of money in overtime that was saved ranged in the thousands of dollars. At that point the system had by far paid for itself.

6 Appendix A: Resources

This paper covers many topics, mostly related the Perl language. More information on them can be found in the following references:

6.1 Files

The files that are referenced in this paper are available online at the Roth Consulting web site:

http://www.Roth.Net/Conference/LisaNT/1999

6.2 Win32 Perl

Win32 ports of Perl come in both binary and source form. The most reliable sources for Win32 Perl are:

- ActiveState Tool Corp., source code and binaries: http://www.ActiveState.com
- The Perl home page, source code: http://www.Perl.com

6.3 Win32 Perl Extensions

Perl is an extendable language allowing users to create extensions that supplement the built in capabilities. Since Perl is also a cross platform language it is not capable of utilizing many of Win32's unique abilities. This is where the Win32 extensions come in.
The following Win32 Perl extensions come standard with Win32 Perl from ActiveState:

- Win32
- Win32::NetAdmin
- Win32::Registry
- Win32::Service

The following Win32 Perl extensions can be found at Roth Consulting, http://www.roth.net/

- Win32::ODBC
- Win32::AdminMisc

The following Win32 Perl extension can be found at Aldo Calpini's web site, http://www.divinf.it/dada/perl/

- Win32::API

The following Win32 Perl extension was authored by Jens Helberg <jens.helberg@bosch.com>. Its source code can be found at ftp://ftp.roth.net/pub/ntperl/others/lanman/lanman_1_01.zip

- Win32::Lanman

6.4 Perl Reference Books

There has been many books written on Perl. Two such books that are especially applicable to this paper are:

- Larry Wall, Tom Christiansen, Randal L. Schwartz, Stephen Potter, *Programming Perl*, Second Ed., O'Reilly & Associates, 1996.
- Dave Roth, *Win32 Perl Programming: The Standard Extensions*, Macmillan Technical Publishing, 1999.

One particularly useful book regarding WinNT administration is:

- Æleen Frisch, *Essential Windows NT Administration*, O'Reilly & Associates, 1998.

7 Appendix B: Scripts

The management system allowed for scripts that would be executed. The order of execution was alphabetical so a desired order could be achieved by changing names.

7.1 Script 1: An installation script.

This script will setup a machine so that it is ready to use the management system. This script will:

1. Add the global "Domain Admins" group to the local "Administrators" group.
2. Configure the Scheduler service to logon with a given userid.
3. Grant the specified userid the privilege to logon as a service.
4. Convert the Perl drive to be NTFS if it is not already.
5. Set permissions on the Perl directory such that Administrators have full control and everyone else has read only access.
6. Copy the Perl tree to the local machine.
7. Update the PATH environment variable to include the Perl \bin directory if it is not already.
8. Remove the "last logged on user" so that the next time the machines logon box is displayed there is no specified userid.
9. Reboot the machine.

This script requires the following extensions:

- Win32
- Win32::API
- Win32::Perms
- Win32::Lanman
- Win32::Service
- Win32::NetAdmin
- Win32::Registry
- Win32::AdminMisc

```
# Install.pl
# ----------
# This is an installation script designed to configure
# a machine for use with the network management system.
#
#   Dave Roth
#   Roth Consulting
#   http://www.roth.net
#

use Win32;
use Win32::API;
use Win32::Perms;
use Win32::Lanman;
use Win32::Service;
use Win32::NetAdmin;
use Win32::Registry;
use Win32::AdminMisc;

$Machine = Win32::NodeName();
$Domain = "MyDomain";
$User = "$Domain\\SchedulerAccount";
$Password = '';
```

```
# Even though it's called the "Scheduler" the proper service name is "Schedule"
$Service = 'Schedule';

$PerlDrive = 'm:';
$PerlPath  = '\perl';
$LogFile   = '\\\\server\perlshare$\install.log';

$REMOTE_PERL_PATH = '\\\\server\perlshare$\perl';
$NTFS_CONVERSION_APP = "convert.exe";
$NTFS_CONVERSION_PARAM = "/FS:NTFS /V";

ConfigGroup( $Domain );
ConfigService( $Service, $User, $Password );
if( ConfigDrive( $PerlDrive ) )
{
    SecureDir( "$PerlDrive$PerlPath" );
    CopyDir( $REMOTE_PERL_PATH, "$PerlDrive$PerlPath" );
    ConfigPath( "$Dir\\bin" );
}
ConfigLastUser( "" );
RebootMachine();
Log( $LogFile );
print "Finished.\n";

sub Log
{
  my( $LogFile ) = @_;

  if( open( LOG, ">+ $LogFile" ) )
  {
    flock(LOG, 2);
    seek(LOG, 0, 2);
    print LOG Win32::NodeName(), "\t", scalar( localtime() ), "\n";
    flock(LOG, 8);
    close(LOG);
  }
}

sub RebootMachine
{
    Win32::AdminMisc::ExitWindows( EWX_REBOOT | EWX_FORCE );
}

sub ConfigLastUser
{
    my( $User ) = @_;
    my $Key;

    if( $HKEY_LOCAL_MACHINE->Create( 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',
$Key ) )
    {
        $Key->SetValueEx( $Key, 0, REG_SZ, $User );
        $Key->Close();
    }
}

sub ConfigGroup
{
    my( $Domain ) = @_;

    $Domain .= "\\" if( '' ne $Domain );

    # Make sure that we add the Domain Admins to
    Win32::NetAdmin::LocalGroupAddUsers( '', 'Administrators', "$Domain" . "Domain Admins" );
}

sub ConfigPath
{
    my( $Dir ) = @_;
    my $RegexDir = "$Dir";
    my $Path;
```

```
    # First check to see if the $Dir is already in the path...
    $Path = Win32::AdminMisc::GetEnvVar( 'PATH' );

    # Prepare $Dir for a regex...
    $RegexDir =~ s/([.\\\$])/\\$1/g;
    if( $Path !~ /$Dir/i )
    {
        # Add $Dir to the system (not user) PATH
        Win32::AdminMisc::SetEnvVar( 'PATH', "$Path;$Dir" );
    }
    return( 1 );
}

sub CopyDir
{
    my( $RemoteDir, $DestDir ) = @_;

    print "Copying files from '$RemoteDir' to '$DestDir' ...\n";

    # XCOPY.EXE will autocreate the destination dir
    `xcopy "$RemoteDir\\*.*" "$DestDir\\*.*" /s`;
}

sub SecureDir
{
    my( $Dir ) = @_;
    my( $Perm, $Result );

    print "Securing the $Dir directory...\n";

    `md "$Dir"`;

    if( $Perm = new Win32::Perms( $Dir ) )
    {
        # Remove *all* entries...
        $Perm->Remove( -1 );

        $Perm->Allow( 'Administrators', FULL_CONTROL_FILE, FILE );
        $Perm->Allow( 'Administrators', FULL_CONTROL_DIR, DIR );

        $Perm->Allow( 'Everyone', READ_FILE, FILE );
        $Perm->Allow( 'Everyone', LIST_DIR, DIR );

        $Perm->Owner( 'Administrators' );

        $Result = $Perm->Set();
    }
    else
    {
        print "  Unable to create security descriptor. Directory is not secured.\n";
    }

    return( 0 != $Result );
}

sub ConfigDrive
{
    my( $Drive ) = @_;
    my %Info;

    print "Configuring the $Drive drive ...\n";

    # Make sure that the drive exists
    if( join( ' ', Win32::AdminMisc::GetDrives() ) !~ /$Drive/i )
    {
        print "  The $Drive drive does not exist.\n";
        return( 0 );
    }

    # Do we have a fixed hard drive?
```

```
    if( DRIVE_FIXED != Win32::AdminMisc::GetDriveType( $Drive ) )
    {
        print "  The $Drive drive is not a local fixed hard drive. Skipping drive formatting.\n";
        return( 0 );
    }

    # If the drive is not NTFS then convert it into NTFS
    %Info = Win32::AdminMisc::GetVolumeInfo( $Drive );
    if( 'NTFS' ne $Info{FileSystemName} )
    {
        print "  The $Drive drive is not an NTFS drive.\n";
        print "  Converting from the $Info{FileSystemName} format...\n";
        @{$Log{ntfs_conversion}} = `$NTFS_CONVERSION_APP $Drive $NTFS_CONVERSION_PARAMS`;
    }

    return( 1 );
}

sub ConfigService
{
  my( $Service, $User, $Password ) = @_;

  my $OpenSCManager = new Win32::API( 'advapi32.dll', 'OpenSCManager', [ P,P,L ], L );
  my $OpenService = new Win32::API( 'advapi32.dll', 'OpenService', [ L,P,L ], L );
  my $CloseServiceHandle = new Win32::API( 'advapi32.dll', 'CloseServiceHandle', [ L ], I );
  my $SCHandle, $ServiceHandle;
  my $ServiceString, $UserString, $PasswordString;
  my %ServiceStatus;
  my $UnicodeFiller = "";

  print "Configuring the $Service service ...\n";

  $Result = Win32::Service::GetStatus( '', $Service, \%ServiceStatus );

  # Service status 1 == Service has stopped
  $Result = Win32::Service::StopService( '', $Service ) if( 1 != $ServiceStatus->{CurrentState}
);

  # Convert the string to UNICODE if needed
  $UnicodeFiller = "\x00" if( Win32::API::IsUnicode() );
  ( $ServiceString = $Service ) =~ s/(.)/$1$UnicodeFiller/g;
  ( $UserString = $User ) =~ s/(.)/$1$UnicodeFiller/g;
  ( $PasswordString = $Password ) = s/(.)/$1$UnicodeFiller/g;

  # The flag 0x00F003F requests to open the service manager with full access
  if( $SCHandle = $OpenSCManager->Call( 0, 0, 0x000F003F ) )
  {
    # The flag 0xC0000000 requests GENERIC_READ and GENERIC_WRITE
    if( $ServiceHandle = $OpenService->Call( $SCHandle, $ServiceString, 0xC0000000 ) )
    {
        my $ChangeServiceConfig = new Win32::API( 'advapi32.dll', 'ChangeServiceConfig', [
L,L,L,L,P,P,P,P,P,P ], I );

        # The 0x00000010 flag represents that the service will logon as a user account
        # AND it will create it's own process (not share process space with other services)
        # The 0xFFFFFFFF flag represents no change to this attribute.
        # The 0x00000002 flag represents the service is to auto start
        my $Result = $ChangeServiceConfig->Call( $ServiceHandle,
                                                 0x00000010,
                                                 0x00000002,
                                                 0xFFFFFFFF,
                                                 0,
                                                 0,
                                                 0,
                                                 0,
                                                 $UserString,
                                                 $PasswordString,
                                                 0 );
        if( $Result )
        {
            print "Granting the $User account the privilege to logon as a service\n";
```

```
            # Grant the service account with the privilege to logon as a service...
            $Result = Win32::Lanman::GrantPrivilegeToAccount( '', 'SeServiceLogonRight', [$User]
);
            if( ! $Result )
            {
                print "  Could not grant the privilege: ";
                print Win32::FormatMessage( Win32::Lanman::GetLastError() );
            }
        }
        else
        {
            print "  Could not modify the '$Service' service: ";
            print Win32::FormatMessage( Win32::GetLastError() );
        }

        $CloseServiceHandle->Call( $ServiceHandle );
    }
    else
    {
        print "Could not open the '$Service' service: ";
        print Win32::FormatMessage( Win32::GetLastError() );
        }
    $CloseServiceHandle->Call( $SCHandle );
  }
  else
  {
    print "  Could not open the service manager: ";
    print Win32::FormatMessage( Win32::GetLastError() );
  }

  Win32::Service::StartService( '', $Service );
}
```

7.2 Script 2: The System management engine.

This script is the main engine. This is the script that is executed once a day. From this script all the other scripts will be loaded and executed.
This script requires the following extension:

- Win32::NetAdmin

```
# SysMan.PL
# ----------
# This is the management system engine script. This is the
# core to the entire management system. This script will
# locate any domain controller and begin running scripts
# from there.
#
#   Dave Roth
#   Roth Consulting
#   http://www.roth.net
#

use Win32::NetAdmin;

$REMOTE_PERL_SHARE = 'perlshare$';
$REMOTE_PERL_DIR   = 'perl';
$REPOSITORY_DIR    = 'scripts';

if( Win32::NetAdmin::GetAnyDomainController( '', '', $Server ) )
{
  my @Scripts, $Script;

  $ScriptDir = "$Server\\$REMOTE_PERL_SHARE\\$REPOSITORY_DIR";
  @Scripts = glob( "$ScriptDir\\*.pl" );
  foreach $Script ( sort( @Scripts ) )
  {
    # We only will process scripts that exist and are files
    next unless( -f $Script );
```

```
    print "Loading and running '$Script'...\n";
    require $Script;
  }
}
else
{
  print "Unable to locate remote repository.\n";
}
```

7.3 Script 3: An ODBC configuration script.

This script installs the specified ODBC driver and configures a data source name so that scripts will have database access. This script will:

1. Install and configure the specified ODBC driver if it is not already installed.
2. Configure a new data source name (DSN) if it is not already configured.

This script requires the following extension:

- Win32::ODBC

```
# _ODBC.PL
# ----------
# This is a management system script designed to check for and
# install the MS SQL Server ODBC driver. It also checks
# for a particular ODBC DSN. If it does not exist it will
# be created.
# THIS SCRIPT assumes that ODBC has been already installed
# on the machine.
# Notice that the script name has been prepended with an underscore.
# This will cause this script to be run before other scripts.
#
#   Dave Roth
#   Roth Consulting
#   http://www.roth.net
#

use Win32::Registry;
use Win32::ODBC;

$Dsn = "ConfigServer";
$ODBCDriver = "SQL Server2";

$DestDir = "$ENV{WINDIR}\\System32";
$REMOTE_DRIVER_DIR = '\\\\server\perlshare$\ODBC\SQLServerDriver';

%DriverList = Win32::ODBC::Drivers();
if( ! defined $DriverList{$ODBCDriver} )
{
  my $Key;

  # Copy all the SQL Server ODBC Driver files...
#  `xcopy "$REMOTE_DRIVER_DIR\*.*" "$DestDir\*.*" /s`;

  # We need to install the SQL Server driver.
  if( $HKEY_LOCAL_MACHINE->Create( "SOFTWARE\\ODBC\\ODBCINST.INI\\$ODBCDriver", $Key ) )
  {
    my $SubKey;

    $Key->SetValueEx( 'APILevel', 0, REG_SZ, "2" );
    $Key->SetValueEx( 'ConnectFunctions', 0, REG_SZ, "YYY" );
    $Key->SetValueEx( 'CPTimeout', 0, REG_SZ, "60" );
    $Key->SetValueEx( 'Driver', 0, REG_SZ, "$DestDir\\SQLSRV32.DLL" );
    $Key->SetValueEx( 'DriverODBCVer', 0, REG_SZ, "03.50" );
    $Key->SetValueEx( 'FileUsage', 0, REG_SZ, "0" );
    $Key->SetValueEx( 'Setup', 0, REG_SZ, "$DestDir\\SQLSRV32.DLL" );
    $Key->SetValueEx( 'SQLLevel', 0, REG_SZ, "1" );

    if( $Key->Create( 'FileList', $SubKey ) )
    {
      $SubKey->SetValueEx( 'CTL3D32.DLL', 0, REG_SZ, "$DestDir\\CTL3D32.DLL" );
```

```
      $SubKey->SetValueEx( 'DBNMPNTW.DLL', 0, REG_SZ, "$DestDir\\DBNMPNTW.DLL" );
      $SubKey->SetValueEx( 'DRVSSRVR.HLP', 0, REG_SZ, "$DestDir\\DRVSSRVR.HLP" );
      $SubKey->SetValueEx( 'MSVCRT40.DLL', 0, REG_SZ, "$DestDir\\MSVCRT40.DLL" );
      $SubKey->SetValueEx( 'SQLSRV32.DLL', 0, REG_SZ, "$DestDir\\SQLSRV32.DLL" );

      $SubKey->Close();
    }
    $Key->Close();

    # Update the installed driver list...
    if( $HKEY_LOCAL_MACHINE->Create( 'SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers', $Key ) )
    {
      $Key->SetValueEx( $ODBCDriver, 0, REG_SZ, "Installed" );
      $Key->Close();
    }
  }
}

%DSNList = Win32::ODBC::DataSources();
# We should probably perform a better test to take into account
# mixed case...
if( ! defined $DSNList{$Dsn} )
{
    if( ! Win32::ODBC::ConfigDSN( ODBC_ADD_DSN, $ODBCDriver, ("DSN=$Dsn", "Description=The
Configuration Database", "Server=dbserver", "Trusted_Connection=Yes", "Database=ConfigLogs" ) ) )
    {
        print "Unable to create DSN: " . Win32::ODBC::Error() . "\n";
    }
}
print "Finished.\n";
# We must return a 1 value to indicate this script was successfully loaded
return( 1 );
```

7.4 Script 4: A script to update a database with computer statistics.

This script will setup a machine so that it is ready to use the management system. This script will:

1. Gather information about the computer, processor and operating system.
2. Gather information about each drive.
3. Update a database with the gathered information.

This script requires the following extensions:

- Win32::ODBC
- Win32::AdminMisc

```
# OS.PL
# ----------
# This is a management system script designed to discover
# information about both the computer and OS. Once the
# information has been obtained it is then set to a database
# where it is stored.
#
# This script assumes that the database has 2 tables:
#   1) Computer:
#       ID          int(), primary key, autoincrimenting
#       Name        char
#       Processor   char
#       Speed       int
#       OS          char
#       ServicePack char
#       Version     float
#       Build       int
#       MMX         int (used as a boolean or bit)
#       Ram         int
#       PageFile    int
#
#   2) Drives:
#       ID          int, primary key, autoincrimenting
#       Size        int
#       Free        int
```

```
#       Drive       char
#       Type        char
#       Computer    int
#
#   Dave Roth
#   Roth Consulting
#   http://www.roth.net
#

use Win32::AdminMisc;
use Win32::ODBC;

$DSN = "Machines";
$Machine = Win32::NodeName();

%DriveType = (
    &DRIVE_REMOTE    => "remote",
    &DRIVE_REMOVABLE => "removable",
    &DRIVE_FIXED     => "fixed",
    &DRIVE_CDROM     => "cdrom",
    &DRIVE_RAMDISK   => "ramdisk",
);

# Get memory info...
%Mem = Win32::AdminMisc::GetMemoryInfo();

# Get processor info...
%Processor = Win32::AdminMisc::GetProcessorInfo();

# Get OS info then fix the servicepack and platform values...
%OS = Win32::AdminMisc::GetWinVersion();
( $OS{ServicePack} ) = ( $OS{CSD} =~ /(\d*?)$/ );
$OS{Platform} =~ s/win32_//i;

# Get drive info...
foreach $Drive ( Win32::AdminMisc::GetDrives() )
{
    my $Type = Win32::AdminMisc::GetDriveType( $Drive );

    if( DRIVE_REMOTE != $Type )
    {
        $Drives{$Drive} = { size => 0, free => 0 };

        if( DRIVE_FIXED == $Type )
        {
            ($Drives{$Drive}->{size}, $Drives{$Drive}->{free} ) =
Win32::AdminMisc::GetDriveSpace( $Drive );
        }

        $Drives{$Drive}->{type} = $Type;
        if( DRIVE_REMOVABLE != $Type && DRIVE_CDROM != $Type )
        {
            my %Volume = Win32::AdminMisc::GetVolumeInfo( $Drive );
            $Drives{$Drive}->{filesystem} = $Volume{FileSystemName};
        }
    }
}

if( $db = new Win32::ODBC( $DSN ) )
{
    $Id = GetDbId( $db, $Machine );
    if( ! $Id )
    {
        $db->Sql( "INSERT INTO Computer (Name) VALUES ('$Machine')" );
        $Id = GetDbId( $db, $Machine );
    }

    if( $Id )
    {
        $Update = "UPDATE Computer SET " .
               "  ServicePack = '$OS{ServicePack}', " .
```

```
                "  Version      = $OS{Major}.$OS{Minor}, " .
                "  OS           = '$OS{Platform}', " .
                "  Build        = $OS{Build}, " .
                "  Processor    = '$Processor{ProcessorType}', " .
                "  Speed        = $Processor{Win32ProcessorSpeed}, " .
                "  MMX          = $Processor{MMX}, " .
                "  Ram          = $Mem{RAMTotal}, " .
                "  PageFile     = $Mem{PageTotal} " .
                "WHERE ID = $Id";
        if( $db->Sql( $Update ) )
        {
          print "Could not update computer info: " . $db->Error() . "\n";
        }

        if( $db->Sql( "DELETE Drives WHERE Computer=$Id" ) )
        {
           print "Could not delete drive info: " . $db->Error() . "\n";
        }

        foreach $Drive ( sort( keys( %Drives ) ) )
        {
          my $Insert = "INSERT INTO Drives " .
                       "(Drive,Size,Free,Type,Computer) " .
                       "VALUES ( '$Drive', $Drives{$Drive}->{size}, " .
                       "$Drives{$Drive}->{free}, '$DriveType{$Drives{$Drive}->{type}}', " .
                       "$Id )";
          if( $db->Sql( $Insert ) )
          {
             print "Could not insert drive $Drive info: " . $db->Error() . "\n";
          }
        }
    }
    $db->Close();
}
else
{
    print "Unable to connect to the database: " . Win32::ODBC::Error() . "\n";
}

print "Finished.\n";

sub GetDbId
{
    my( $db, $Computer ) = @_;
    my( %Data );
    if( ! $db->Sql( "SELECT DISTINCT * FROM Computer WHERE Name like '$Computer' " ) )
    {
        if( $db->FetchRow() )
        {
            %Data = $db->DataHash( 'ID' );
        }
    }
    return( $Data{ID} );
}
# We must return a 1 value to indicate this script was successfully loaded
return( 1 );
```

7.5 Script 5: A script to report event log warnings and errors.

This script will setup a machine so that it is ready to use the management system. This script will:

1. Gather information from the local event log.
2. Update a database with the gathered information.

This script requires the following extensions:

- Win32::ODBC
- Win32::EventLog

```
# EVENTS.PL
# ----------
# This is a management system script designed to discover
# eventlog errors and warnings. Such data is procured and
```

```
# submitted to a database where it is stored.
#
# This script assumes that the database has 2 tables:
#   1) Computer:
#       ID          int(), primary key, autoincrimenting
#       Name        char
#
#   2) Events:
#       Source      char
#       Event       char
#       Type        int
#       Userid      char
#       Computer    int
#       Time        datetime
#
#   Dave Roth
#   Roth Consulting
#   http://www.roth.net
#

use Win32::EventLog;
use Win32::ODBC;

$DSN = "Machines";
$Machine = Win32::NodeName();

$SecPerDay = 24 * 60 * 60;
$Now = time();
$TimeLimit = $Now - ( $SecPerDay * 1 );

if( $db = new Win32::ODBC( $DSN ) )
{
  my $Id = GetDbId( $db, $Machine );
  if( $Id )
  {
    foreach $Source ( 'System', 'Application', 'Security' )
    {
      my $Event = new Win32::EventLog( $Source, $Machine );
      if( $Event )
      {
        if( $Event->GetNumber( $Num ) )
        {
          my %EventData;

          $Flag = EVENTLOG_BACKWARDS_READ | EVENTLOG_SEQUENTIAL_READ;
          do
          {
            if( $Event->Read( $Flag, $Num, \%EventData ) )
            {
              # Escape all ' chars for the sake of SQL syntax...
              map{ $EventData{$_} =~ s/'/''/g; } ( keys( %EventData ) );

              # We are only going to log errors...
              if( ( $EventData{EventType} & EVENTLOG_ERROR_TYPE )
                  || ( $EventData{EventType} & EVENTLOG_WARNING_TYPE ) )
              {
                my $Insert;
                my @Date = localtime( $EventData{TimeGenerated} );
                my $DateStamp = sprintf( "{ts '%04d-%02d-%02d %02d:%02d:%02d' }",
                                         $Date[5] + 1900,
                                         $Date[4] + 1,
                                         $Date[3],
                                         $Date[2],
                                         $Date[1],
                                         $Date[0] );
                my $Insert = "INSERT INTO Events " .
                             "(Source, Event, Type, Userid, Time, Computer) " .
                             "VALUES ('$EventData{Source}', '$EventData{Event}', " .
                             "$EventData{EventType}, '$EventData{User}', " .
                             "$DateStamp, $Id )";
                if( $db->Sql( $Insert ) )
```

```
                {
                  print "Unable to insert event data: " . $db->Error() . "\n";
                }
              }
            }
            else
            {
              undef %EventData;
            }
            # This will cause the next reading of the registry to move to the
            # next record automatically.
            $Num = 0;
          } while( $TimeLimit < $EventData{TimeGenerated} );

          Win32::EventLog::CloseEventLog( $Event->{handle} );
        }
      }
    }
  }
  $db->Close();
}
else
{
  print "Unable to connect to database: " . Win32::ODBC::Error() . "\n";
}

sub GetDbId
{
    my( $db, $Computer ) = @_;
    my( %Data );
    if( ! $db->Sql( "SELECT DISTINCT * FROM Computer WHERE Name like '$Computer' " ) )
    {
        if( $db->FetchRow() )
        {
            %Data = $db->DataHash( 'ID' );
        }
    }
    return( $Data{ID} );
}
# We must return a 1 value to indicate this script was successfully loaded
return( 1 );
```

State Driven Software Installation for Windows NT

Martin Sjölin
*Warburg Dillon Read (WDR)**
P.O. Box, 8098 Zürich, Switzerland
martin.sjoelin@wdr.com

Abstract

We have implemented a state driven installation mechanism to simplify the installation of software under Windows NT. We have a "central configuration database" which defines the target state of the machines, e.g. a declarative definition instead of an operational definition. We describe the procedure used to install the packages and some related issues concerning software delivery; software configuration; and the actual software installation for workstations components and user components. Partial details of our implementation of System V packaging is included in the Appendix.

1 Introduction

To make the installation of software in the desktop computing environment of UBS, we have implemented a state driven software installation mechanism. We are using Microsoft NT workstation as the standard desktop with Netware servers or NT servers as the associated file and print (f&p) servers or as home servers for user data. The tools described have been productive use since the late 1997.

We have selected to "lock" down the desktop and the machine for the normal users. And having a standard set of applications which are "packaged" using our own packaging format similar to the "System V" packaging [SysV]. This format is used for delivery, configuration and installation of the applications (please see the appendix for more details). The packages are delivered via CD or SMS [SMS1.2] to servers on the local LAN in the branch office. Then is the software installed on the f&p servers and/or workstations (clients).

To ensure that all workstations have a standard set of applications installed, we store the configuration of each workstation in a "centralized database" [1] (this is not an inventory database). The configuration data defines pro workstation which packages should be installed, the installation mode, and the installation order between the packages. The users of the workstation is restricted to the set of packages listed in the configuration database for the workstation.

At well defined times, a NT service running on all NT workstations and NT servers wakes up or is awakened. The service compares the current state of the machine (which packages are actually installed) against the state defined the configuration data. The service then performs the necessary actions to reach the state as defined in the configuration data.

When the users login into "their" workstations, we compare the user components already installed into the user's profile (including home drive) against the set of packages installed on the workstation itself. We compute the difference between the state of the workstation and user's profile, and then perform the necessary actions to update the user's profile to the state of workstation[2]. We name this *flex seating* since the users can flex or change between the different workstations which are similarly configured, having access to all the standard application via their profile[3].

*The work described was done while the author was at UBS AG, Private and Corporate Consumer Division, between 1996 and 1998.

[1] The database is stored securely on a server, either as flat files, ini files, or in a RDBMS.

[2] In reality, it is more complicated, but the scope of the paper does not allow us to elaborate on the software authorization mechanism.

[3] To allow full roaming between different branch offices or different resource domains, we need to add support for on demand package installation.

For the NT servers, the same process applies as for the NT workstations, but with a single extension - the NT servers must also handle packages which are exported to the clients, e.g. dictionaries for Microsoft Office. Compare this against the packages which are installed for the NT server itself, such as SNMP, backup, virus scanner etc.

As seen from the above description, we have a declarative definition of what should be installed on each machine and by extension which application users have access to. By have the state of the machine defined in a configuration database, it is very easy to recreate the state of the machine after a crash[4]. Compare this with more "normal" operational definition where the state of the machine is defined by the set of application installed on the machine, possible together stored a central inventory of the machines.

2 Computing Environment

We support user authentication against a NT domain with home directory and profiles stored on a NT server, or against Novell's NDS where the user profile and home directory is stored on a local Netware 4.x server. For the typical environment in a branch office (the listed server are logical servers, often there will be a single physical server), we have:

- *Clients*, the workstation or desktop machines, which are all running Microsoft NT 4.0. All client have an associated shared application server.
- *Shared Application Server* which is where common components like dictionaries, templates, and seldom used applications are stored. This is often the print server for the associated clients. A few standard SMB shares are available for the client.
- *Home Server* is where the user home directory is mapped and stored together with the NT 4 user profile.
- *Package Server* stores the packages to be installed. Exports a SMB share with the *package spool area*. For a SMS distribution server we also have the standard SMS package share. For NT servers, we distribute package via SMS and for Netware server we distribute packages via monthly CDs. Or via HTTP or FTP in emergency cases.
- *Configuration Data Server* stores the "configuration database" as `ini` style file for the machines.

[4] We use the same mechanism for the initial setup of the machine by providing a small set of bootstrap tools in the OEM directory of the Microsoft unattended setup, and then invoking our installation process.

3 Implementation

The software installation (including both removal old versions and additions of new versions of software) can logically be split into the installation of:

- F&P servers components is the shared context of packages. These are made available for the workstation from its associated f&p server via SMB shares. For NT servers, the software installation is done by the "EUP Installer Service" (`eupsrv`).
- Workstation components includes the workstation context of a package for a *shared* mode installation, but can also include the shared context and the workstation context for the *local* mode installation. As for the NT server, the software is installed by the "EUP Installer Service" (`eupsrv`).
- User components are only the user contexts of a package, installed by by the `euplogon` program at user login time.

We treat the installation of server components and workstation components as a single case. From the actual calculations the server case is an extension of the workstation case to handle the shared context exported to the clients via the SMB shares.

For the installation of the server and workstation components, we need the configuration data describing the wanted or target state of the machine. In the configuration data for the machines are stored: the list of packages to be installed, their installation mode (local, shared, or exported), the location of the package spool, and control parameters for `eupsrv`. The installation order of the packages is the order of the packages in the configuration data.

3.1 Machine Components

The first step for the `eupsrv` is to determine if any users are logged in - we user several methods: a GINA [GINA], enumeration of open desktops, and a flag set by the `euplogon` at user login. If an user is present, we either present a warning dialog asking the user to logout as soon as possible, or we will retry within a pre configured time limit (defaults to 4 hours). Once the `eupsrv` can detected no users, it optionally login into the associated shared application server using the credentials provided in the registry configuration. This is to get access to the configuration data, to the shared application server, and to the package server.

The initial step is to determine the set of packages already installed on the machine, state (installed or removed etc.), mode (local, shared or exported), the installation order, and who installed them and when. By scanning the pkginfo directory, for a NT workstation and a NT server: `%SystemRoot%\Config\PkgInfo\`. And for the NT server the list of exported packages: `\\server\PkgInfo\`.

For each package we read the installed package's `pkginfo.ini` and `pkgvars.ini` files to retrieve the installation time, the shared application server, installation status (successful installed, partially installed, or removed):

```
[Status]
PSTAMP=MARTIN980226113655
UserId=SYSTEM
Status=Successfully Installed
Date=1999.11.11:23:05:00
```

The shared application server associated with each package is retrieved by reading the value from the `pkgvars.ini` file. The installation mode is *shared* if no shared context have been locally installed. If a shared context and workstation context have been locally installed, the package have been *locally* installed.

Thus, we have determined the set of current packages installed on the machine:

$$CurrentState = set\ of\ packages\ already\ installed \quad (1)$$

We connect to the configuration database to retrieve the target state, which is an ordered list of packages with their installation mode:

$$TargetState = set\ of\ packages\ in\ the\ configuration \quad (2)$$

Having *CurrentState* and *TargetState*, we compute the set of package necessary to remove from the machine, by taking all packages not present in the *TargetState* but currently found on the machine, *CurrentState*:

$$Pkgs2Remove = CurrentState \setminus TargetState \quad (3)$$

By default, we filter out package which have not been installed by the service from the *Pkgs2Remove* set, since removing the Emacs package which a developer have installed for his own use is not acceptable.

The next step is to compute the set of packages which we must install on the machine to reach *TargetState*, which is the set of all package present in the *TargetState* but not in the *CurrentState*:

$$Pkgs2Install = TargetState \setminus CurrentState \quad (4)$$

Notice that two entries when comparing are only considered equal if the following conditions are met:

1. Same package name (`ubsperl_5_un_1.4`),
2. Same installation mode (shared, local or exported),
3. Same shared application server for *shared* installation mode, and
4. Already installed package is installed successfully.

Once we have computed the *Pkgs2Install* and *Pkgs2Remove* sets, we order the *Pkgs2Remove* in the reverse installation time to avoid any problems with install time or run-time dependencies. The *Pkgs2Install* set should will the same order as specified in the configuration data. If both *Pkgs2Remove* and *Pkgs2Install* sets are empty we are finished.

Before we start the actual package addition or package removal, we verify for all package in the *Pkgs2Install* set that the package is actually present on the package server. We also verify that for shared mode installation that the correct version of the shared context of the package is installed on the the shared application server. If not both of these conditions are fulfilled, we will ignore the package during the actual installation.

We start traversing the *Pkgs2Remove* set and remove the already installed packages by calling `pkgrm`. For the shared mode, we remove the workstation context (which is the only context installed). And for the local mode, we first remove the workstation context followed by the shared context.

The following step is to process the *Pkgs2Install* set and to install the new packages by invoking the `pkgadd` for each package. The major parameters are: package name, shared application server, and path to package spool area. For locally installed packages, we first install the shared context and then the workstation context.

The `eupsrv` invokes *pre-* and *post-* scripts stored on the machine in a secure location and stored on the shared application server before the enumeration of the installed package as well as after adding the last package. This enable packagers or the local supervisor to modify the state of the machines and have to influence what are installed.

Last, for a package which requires an immediate reboot, as specified by the restart flag in the `pkginfo.ini`, the `eupsrv` will force a reboot of the machine once the package have been removed (or added). For packages which requires a reboot before becoming operational, the `eupsrv` will reboot after the last action (install or remove) has been done. The `eupsrv` will continue operations after the reboot, either being restarted by the SMS PCM (Package Command Manager) or by itself.

3.2 User Components

When the user login to a NT workstation, the `euplogon` program is invoked - we have replaced the standard `UserInit` registry value. The `euplogon` program compares the list of packages already installed on the machine (by scanning the machine's pkginfo directory) against the list of user contexts already installed to the user profile and home drive (by scanning the pkginfo directory stored on the user home drive and checking the cached values in the registry).

Using roughly the same computation as described in the previous section, the program determine the set of user contexts to remove and the set of user contexts to add to ensure that the user have the same set of packages as already installed on the machine.

One minor change is that we only remove a user context if the current machine where the user is logged in to is the same machine where we originally installed the user context. We avoid removing a package when an user temporarily login into another workstation than the her normal workstation. Instead, we hide any shortcut in the user program menus by setting the `HIDDEN` bit on the shortcut.

4 SMS Integration

Since one year, we support software distribution via SMS and also initiate software installation via SMS for the "pure" NT environment. This integration caused a number of changes or improvement in the `eupsrv`.

4.1 Push versus Poll

When we control software installation using SMS, we would like to initiate the software installation via a standard SMS job. This changes the operational mode from "poll" (check if configuration data have changes at regular intervals) to "push" (start now and verify if the state of the machine matches the state as defined the configuration data).

The standard operational mode of the `eupsrv` in the initial release was to poll the configuration data (a `ini` file) every fourth hours when a user was not logged in. To avoid re reading the file when it not have changed, we cache the last modification date and size in the `eupsrv` and compare those attribute to determine if the file have changed.

We have the `PCM` installed on all NT workstations, NT servers, and SMS distribution servers as a service. We extended the `eupsrv` to be started from

the PCM and no longer polling the configuration data at regular interval. When called from a SMS job, the **eupsrv** service is started from the PCM. The command line executable blocks until the **eupsrv** service returns with a status code before it exits (and thus blocks the PCM and the current SMS job).

The **eupsrv** was extend with three command line option to support:

- **eupsrv -run** which verify the state of the machine against the state as defined the configuration data. This command is invoked as part of a SMS job to force an update of a machine.
- **eupsrv -pkgadd** *package-id,...* to enable the installation of one or more package from a SMS job, *if* the packages are included in the configuration data for the machine. The installation mode is read from the configuration data.
- **eupsrv -pkgrm** *SMS-ID,...* to enable the removal of one or more packages from the target machine. No check is done for run time dependencies, so this can break an existing installation.

Some complication arise in the handling of packages which requires a reboot or who should restart the **eupsrv** after the reboot. The simple solution was to stop the SMS PCM service when a reboot was required, and having **eupsrv** forcing a reboot of the machine. After the reboot the PCM detects that the SMS job was not finished, and retries SMS job including the **eupsrv** command.

4.2 SMS Software Delivery

One major problem with SMS 1.2 is that the actual software distribution is not atomic to the distribution servers. By atomic delivery, we mean that either is the package on the SMS share to 100% or to 0%, not partially present.

To all SMS distribution jobs, we added a small batch script which created a flag file, in the root of the package directory once the package was delivered to the distribution server. The second step was to extend the **eupsrv** to verify if the package was fully delivered to the share by checking for the flag file.

4.3 Shared Contexts Coordination

For packages which are installed in shared mode on the workstations, we must ensure that the shared context is present on the associated shared application server and also that the correct version of the package is present. This was added as part of the SMS extensions to ensure that SMS jobs sent directly to the workstations did not try install the workstation context of the package before the shared context was present on the associated f&p server.

5 The Good, the Bad and the Ugly

We have learned a few lessons during the last years during the development of the installation mechanism, especially issues which crept up during the integration with SMS. What follows is a partial lists of points.

We have discovered bugs in both Microsoft Networking code and the Netware client for NT. The network and security issues to get the **eupsrv** working correctly have been causing headaches.

The "Poll" mode sounds good and looks good, but the local supervisor needs more control over when an software installation is started, which workstations should be done, and a mean to reduce the load on the network and/or the server. We solved this by adding a common configuration file on the local Netware server, but ...

Under Netware 4.x, we need to authenticate against NDS before we can read files on the server, e.g. the package or configuration files. In the case of the configuration file above, even though we had limited the number of parallel updates to, say 5, we overloaded the NDS authentication server when all the workstations tried to read the configuration file. For the Netware environment, a possible solution would to use a TCP server for common configuration data, or start the **eupsrv** via Netware Workstation Manager.

The current general policy for the installation of a new version of a package is first to completely remove the old version and then install the new version. But most of the new versions of packages are incremental improvement or small changes to the components. The current approach causes a

lot of extra network traffic by coping data from the package server which to a large extend was already present on the target machine. Either overwrite functionality or a delta package mechanism to reduce the network traffic as well as the installation time should be considered. The current "distributed" database of installed components, `pkgstat.dat`, in each package's pkginfo directory pro context, should be centralized to easy the implementation of overwrite packages.

Centralize the configuration data in a central repository (database) and distribute it using LDAP [LDAP] or similar. By extending the initial design with new configuration files on the shared application server, we added more and more configuration data in several location instead of going for a unified interface. With the new release using the *EUP Values* we have initiated the centralization and collection of the configuration data into a single location and single interface.

During the last year, we have put a lot of work into the creation of standard and guidelines for how to create "clean" packages, package creation tools, and package verification tools (against a central package database). This is absolutely essential to improve the quality of released packages and get a stable desktop platform.

For the future, with Office 2000 [Office2000] as well as NT 2000, Microsoft Software Installer [MSI, MSIT] (MSI) is looming on the horizon together with new release of InstallShield [InstallS] as well as SMS 2.0. We have started working on how we can replace part of our in-house developed components and integrate them with MSI.

Notice that the ideas expressed with a state driven installation can be realized using any installation format as long as the state is saved on the target system and in the user registry and/or home drive. There must also exist one-to-one mapping between a package version and the stored state to enable us to uniquely determine which version was installed.

6 Acknowledgments

The work described is the work of a lot of people over the last four to five years at UBS, and an incomplete list: Andre Aeppli, Martin Hufschmid, Diedrich Klarmann, Peter Kurz, Nick Riordan, Martin Schaible, Jeffrey Tolmie. And especially thanks to those I have forgotten.

References

[SMS1.2] R. Anderson, J. Farhat et al, *Microsoft SMS 1.2 Administrator's Survival Guide*, Sams Publishing, (1997).

[InstallS] *InstallShield for Windows Installer: Overview, White Paper*, Version 1.0, December (1998).

[LDAP] T. Howes and M. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, Macmillan Technical Pub., (1997).

[MSI] M. Kelly, *Gain Control of Application Setup with the New Windows Installer*, Microsoft Systems Journal Sept (1998) p. 15-27.

[MSIT] *Microsoft's Software Installation Technology. Part 1: Client-side Installation Service*, Directions on Microsoft (Research), March (1998).

[Office2000] *Microsoft Office 2000 Deployment and Maintenance, White Paper*, `http://www.microsoft.com/office`

[SysV] *System V Packaging Manual*, `http://docs.sun.com`

[Win5] *WinInstall version 5.1*

[GINA] *Winlogon User Interface*, Microsoft Win32 Software Development Kit for Microsoft Windows.

A A System V Style Packaging System for Windows NT

The installation mechanism described is build on top of a packaging system for Windows NT which have been implemented in house since the middle 90's.

The application format is an implementation of a "System V" [SysV] like packaging system as under Solaris/SunOS, with improvements to enable a

smoother integration with both the Windows NT and the Netware computing environment. We used concepts (`prototype` file, `pkgmap` file, etc.) from the System V packaging and also integrated features (variables and variable expansion in the output files) from the WinInstall [Win5, InstallS] product. We have kept the name of the standard packaging tools: `pkgmk` for package creation, `pkgadd` for package installation, `pkgchk` for package verification, and `pkgrm` for package removal.

We will try to give an overview of our packaging system as used in the environment with Windows NT clients connected to f&p servers. The target is to describe enough of the packaging system to make the installation mechanisms clear, without going into the more esoteric details of the actual implementation.

A.1 Background

Why packaging? Why not simply use WinInstall [Win5], InstallShield [InstallS], or the format as provided by the Microsoft SMS Installer [SMS1.2]?

When the Windows NT 4.0 project started in 90's, the existing technology was not good enough, and did not support both Netware servers as well as NT servers as target for application installation or parts thereof. We must support application installation to both server platforms and we had already a working packaging system for Netware servers. We also have experience with the System V packaging tools which we have extended extensively.

As installation targets, we must support NT workstations, NT servers, and Netware servers with a single packaging format for all platforms. We have a mixed environment where NT workstations can have a f&p server which is a NT server or a Netware server. And for the applications installed on the workstations, there must be no difference if the associated f&p server is a NT server or a Netware server.

Further, we must also support different languages on the server (English, German, French and Italian); flexible directory structure (not all application are installed into `C:\Program Files`; co-existing of 16-bit and 32-bit applications; configuration of application (e.g. where is the SQL server); and clean removal of applications.

A.2 Shared, Workstation and User Contexts

We generally talk about three different parts of a package: the shared context, the workstation context, and the user context. Before we go into details about the different contexts, we need to mention that a package can be installed in three modes:

Local mode is when everything in the package is installed on the target machine.

Shared mode is when part of the package is installed on the f&p server associated with a workstation. The typically installed components are dictionaries, help files, clip arts, or shared database files. In this case a set of workstations "share" the common items on the server.

Exported mode is only valid for a NT server (a shared application server) where a package is "exported" to a set of a workstations. The shared context of the package is installed on the server, but read/referenced by the clients.

It is important to notice that the package itself can be created (and should) in such a way to support both local and shared installation modes. Often the installation mode is selected at *installation time* and not at package creation time.

The *shared context* of a package are those files which are part of the shared installation. This can only be files and no registry entries, since the files are made available to the client via a SMB share[5] on the associated f&p server.

The *workstation context* of a package are file components and registry components which are installed on the client itself. The registry changes are made to the machine hive (HKLM[6]), often adding configuration information for the application or adding definitions for a DLL. The files are typically installed on the machine either in the `%SystemRoot%`, `%System32%`, or under `%SystemDrive%` directory. Typically a shared DLL (mfc42.dll) goes into %System32% and netscape.exe into `%SystemDrive%\ie_appl\ie_4\netscape`

[5] Typically `ie_appl`

[6] Hive Key Local Machine

The *user context* of a package contains the registry changes to the user hive (HKCU[7]), possible configuration files into the user home drive, or changes to the user profile. An typical example is to add a shortcut to the program menu, pointing to the application installed in the workstation context of the same package.

A.3 Packaging Classes

As under System V packaging, we provides a set of standard classes to perform the usual manipulations needed to install an application. All but one class is external, that is implemented as an executable outside of `pkgadd`:

none used for directory creation, file copying and also to copy the `install.txt` and the `pkginfo.ini` into pkginfo directory (internally implemented class).

registry class to do changes to the registry (`REG_SZ`, `REG_DWORD`, `REG_MULTI_SZ` etc.).

iniclass for changes to `.ini` files.

execbat to invoke `CMD` script during package add or package removal.

pkgenv to change the machine or user environment.

pkgpath to add or remove components to the machine (system) path or user path.

shortcut to create shortcut to an executable in the user's start menu or in the default/all menu for the machine.

pkgassoc to add an association between a file type (`.htm`) and an executable (`netscape.exe`).

pkghosts to add an hostname entry to `etc\hosts`

pkgserv to add a service definition to the `etc\service`

regocx to register a "self registration" DLL.

resolve to replace packaging variables in the input file with variable values (variable expansion).

[7] Hive Key Current User

In System V packaging, we have the concept of classes which are used to determine which part of a package is going to be installed. We do not have classes to do dynamical installation configuration, instead it is possible to place conditionals in the `prototyp.txt` as for the C pre processor (`#if... #else... #endif`). The conditional allow the package creator to query different package variables to determine which files to install. A typically examples is to install the correct sound card drivers dependent on the type of the machine.

A.4 Packaging Variables

In System V packaging, the `pkginfo` file is used together with the `request` script to gather the input necessary for the correct installation of the package. The environment created by the output from the `request` script can then be used during the `preinstall` and `postinstall` scripts.

In our implementation, we do not support interactive prompting (à la `request` script) for the install time configuration. All information must either be present on the machine at installation time, it must be possible retrieve from a configuration file, or it is possible to compute the configuration information.

We have the `pkginfo.ini` file. The following is an example from the standard UBS Perl package:

```
[PKGINFO]
PackageName=UBSPerl
PackageVersion=1.4
Description=Perl-Win32
ProgramName=perl
ProgramVersion=5
ProgramVendor=GNU
VendorVersion=5.004p2
ProgramLanguage=UN
PackageCategory=1
PackageType=0
PackageArchitecture=W32
InstallType=FULL
TargetArchitecture=LAN,STD
DiskSpace=6729603
DiskUser=0
DiskShared=0
DiskWorkstation=6729603
Restart=No

[R-Dependencies]
RT_MFC>=4,UN

[I-Dependencies]
RT_SYSTEM=4,IE

[Status]
PSTAMP=MARTIN970814135523
```

The variables listed in the section 'PKGINFO' are available during the whole packaging installation

process. Further, for each machine, we have a packaging variable configuration file, `pkgvars.cfg`, which contains the standard mapping between the defined official variable names and the locations. This file also define the standard menu name and structure, the standard protection code for shared, workstation and user files. As an example for a Netware server with international English release:

```
System32        = "%SystemRoot%\system32"
UserRoot        = "%UserDrive%"
PackageData     = "%UserRoot%\%PackageName%"
AllUsers        = "%SystemRoot%\Profiles\All Users"
UserDrive       = "H:"
PackageDir      = "%ApplRoot%\%PackageID%"
UserMenu        = "%UserProfile%\Start Menu"
StartupMenu     = "%ProgramMenu%\Startup"
ReleaseDir      = "IE_APPL"
ProgramMenu     = "%UserMenu%\Programs"
DeveloperMenu   = "%ProgramMenu%\Development Tools"
DocumentsMenu   = "%DeveloperMenu%\Documents"
OfficeMenu      = "%ProgramMenu%\Office Applications"
PersonalMenu    = "%ProgramMenu%\Personal"
ApplDrive       = "I:"
ApplRoot        = "\\%server%\SYS2\%ReleaseDir%"
```

By having different configuration files for NT servers and Netware servers, we hide the differences in the directory structure. We also define in the guidelines which variables can be used and in which context, e.g. in the shared context your are not allowed to install a DLL into **%System32%** directory. You can only install a DLL into this directory in workstation context.

The variables can be used in the `prototyp.txt` and in all the files which are input to the external classes. By using variables in the file input to `registry` class, we will have the correct path to the executable:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\\Perl]
"PERL5LIB"="%SystemDrive#%\\UBS_Tools\\%ProgramName#%\\lib;"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.pl]
@="Perl"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Perl\Shell\Open\Command]
@="perl.exe %1 %*"
```

So far, the packaging variables described only have allowed configuration based on the static settings stored on the target machines, using the configuration files and the installation target. We have extended the packaging system to allow installation time querying of variables values, so called *EUP Values*. The actual value of the variable can be a single string value or multiple values. All variables which begin with a standard prefix are queried at installation time by `pkgadd` via an well defined interface exported by a DLL[8].

[8] By exchanging DLLs, we can have different data sources and transport mechanisms.

A.5 pkgmk

As under System V, the main input to the `pkgmk` is the `prototyp.txt` file which describes the components as well as into which context the different components are to be installed:

```
[Shared]
d none "%PackageDir%" ? ? ?

[Workstation]
!search ".\install"
e execbat "preTask.cmd" ? ? ?
!search ".\system32"
e registry "wks.reg" ? ? ?
f none "%System32%\nsrt2432.acm" RO ? ?
c #if %EUP_MachineType%=SERVER
  f none "%System32%\rt32cmp.dll" %WksFileAttr% PD ?
c #else
  f none "%System32%\rt32dcmp.dll" %WksFileAttr% P ?
c #endif
e execbat "postTask.cmd" ? ? ?

[User]
!search ".\install"
e registry "user.reg" ? ?  D
e shortcut "shortcut.sct" ?  ?  ?
```

The `pkgmk` generates a spooled package in a spool area with a standard directory layout. Included is the file `pkgmap.dat` describing the contents of the package which corresponds to the System V `pkgmap`.

A.6 pkgadd

For the installation of a package, or rather a context of package, we use `pkgadd`, specify the full package name, the package spool area, the target machine, and the selected context. Before we install the workstation context we must have a shared context installed, either locally on workstation or the associated the f&p server. This also applies for the user context, which cannot be installed until the workstation context have been installed.

Under System V packaging, we have the `preinstall` and `postinstall` standard scripts which we can use to prepare and/or for the cleanup of the package installation. This will have to be implemented via `BATCH` scripts located in the `prototyp.txt` in the first or last position within in each contexts. For example, see the `postTask.cmd` and `preTask.cmd` in the example `prototyp.txt`.

Under System V, the installation of a package modifies the `/var/sadm/install/contents` to add the

newly installed components and their protections together with the owner of the components. Also, a reference count of the packages which have installed the same components are kept in the `contents` file.

Instead of having a central database (the `contents` file), we selected to have a distributed state pro package and pro context in the local file system on the target machine and in the user home drive/ registry (part of the user profile). Notice that the same format and contents is used for all contexts and all targets. When the package is installed, `pkgadd` creates a "package information" (pkginfo) directory in the proper location on the target (machine and context) using the package name provided as argument to `pkgadd`. For a shared installed package, we will have a pkginfo directory for the shared context on the server; a pkginfo directory for the workstation context on the NT workstation; and a pkginfo directory in the user home drive and registry. Typically, the different locations for the example package `ubsperl_5_un_1.4`:

- `\\zhsv13664\PkgInfo\ubsperl_5_un_1.4\` for the shared context installation on a NT server.
- `\\zhbdvf02\Sys2\PkgInfo\ubsperl_5_un_1.4\` for the shared context installation on a Netware server.
- `%SystemRoot%\Config\PkgInfo\ubsperl_5_un_1.4\Shared\` for the shared context installed on a workstation (local install).
- `%SystemRoot%\Config\PkgInfo\ubsperl_5_un_1.4\` for the workstation context[9].
- `%HOMEDRIVE%\Sys32\PkgInfo\ubsperl_5_un_1.4\` for user context of the package[10].

In the pkginfo directory, we have the `pkginfo.ini` file, which is the same as provided in the package spool, except the information about the installation have been appended (when was the installation done, status, and who did the installation).

Further, we find the `pkgvars.ini` file, which contains the actual packaging variables, and their values, used for the installation of the package. We also have the `pkgstat.dat` file which contains the list of components installed in actual installation order.

All files which are used as input to any of the classes are also kept (e.g. input to *registry* class, `path` class etc.). Notice that the input files in the original package is processed and all the package variables are replaced with their values before the files are feed to the classes. By keeping the input file to the classes, we will not need the package spool during the package removal. Second, we ensure that the same values for the packaging variables are used for the package removal as were used for package installation.

A.7 pkgrm

For the removal of a package, we have `pkgrm` and all the necessary information for removal is present on the machine in the pkginfo directory. The package contexts for a single package should be removed in the reverse installation order, e.g. for a *local* installation the workstation context should be removed before the shared context. Notice that we should not remove the shared context installed on a server until all associated workstation have removed their workstation context. User contexts, being the last installed context for a package, can be removed at any time.

The `preremove` and `postremove` supported by System V have to be replaced with the `PreTask.cmd` etc. Notice that the installed components within the installed package are removed in reverse installation order using the `pkgmap.dat` file stored in the pkginfo directory.

[9] The location is stored in the registry on the machine and set when the machine configured. No hard coded paths are stored in tools themselves

[10] The exact path is stored in user registry and is initialized the first time `pkgadd` is called.

NFS and SMB Data Sharing Within a Heterogeneous Environment: a real world study

Alan Epps
Tektronix
Dr. Glenn Bailey
Tektronix
Douglas Glatz
Tektronix

Abstract:

A common problem encountered in a heterogeneous computing environment which includes both Unix & PC/Windows hosts is sharing data between the different operating systems. Any approach must take into account the different methods of authenticating users, file permissions, and network protocols. With 2000 PC desktops & 800 Unix users employing a variety of NT & Unix servers, the Unix support group at the Color Printing and Imaging Division (CPID) of Tektronix needed a robust system that was inexpensive, easy to administer, simple and effective to use for both PC and Unix users. Administering separate installations of local NFS clients on the PCs had proven to be problematic, causing us to look at a centralized server-based solution that could provide native PC file sharing via the SMB protocol suite. The potential solutions we looked at were: TotalNet by Syntax (sold by Sun as SunPC), Samba v1.9.18p10, NetServices v.1 & v.2 by Auspex, Network Appliance F230 series servers, and Sun's SunLink Server software v1 & v1.1. We compared performance, ease of use by end users, ease of administration, cost, support, training, scalability, and ease of integration into our current environment. As of this writing the conclusion was to use Samba.

1. Introduction:

The current trend of adding NT to already existing Unix environments, which has been gaining steam for the last two to three years, shows little sign of abating. The marketing machine that Microsoft has turned toward the back room server environment has propelled this trend at ever increasing speeds, making significant inroads into the traditionally Unix areas of file services, print services, web services, and even proxy and routing services. Discussions of 'easy of use' and 'initial training time' have had an impact on the management chains throughout our industries, driving Total Cost of Ownership (TCO) decisions from the initial purchase and setup side of the equation without always looking at 'impact costs' following deployment of a particular package. Support issues surrounding a heterogeneous environment of PCs in a Unix environment are often one of the impact costs that is lost in TCO calculations.

Due to increasing system diversity one of the larger impacts on our support organization has been the difficulty of file sharing between the Unix and PC platforms transparently from a user perspective. There are a large number of file sharing solutions available, from local NFS client software installed on the PC to SMB/NFS gateway software installed on a central server, to servers specifically designed to share files between the differing platforms. Traditionally Tektronix has used a locally installed NFS client on each PC, but as the number of PCs grew, and the number of operating systems grew, the difficulty in maintaining current versions, patches, and configurations on all of the machines became unwieldy. As this condition grew worse we began to look at the centralized SMB/NFS gateway designs, both from a pure software solution as well as from a dedicated designed server system.

The industry literature is full of test results and specifications for products and systems all claiming to be better than their competitors. All of these results use the optimal conditions for the particular product under test, maximizing its strengths and minimizing it weaknesses. While the numbers generated by this type of testing are interesting as a optimal end-point exercise, they are rarely indicative of the real world performance one can expect. Due to the need to justify any expenditures we made to upper management, any numbers we presented to them needed to be reproducible once we had

the proposed system in place. As a consequence our testing was done using production machines, hooked into production networks, during the normal working day. While this introduced a large number of potential impacts on the test over which we had little to no control, by running each test multiple times and averaging the resulting numbers we ended up with an aggregate that should be reproducible once the system is put into full production. The test environment breaks down into three discrete elements: the test driver, the network configuration, and the platform under test.

2. Test Environment:

2.1 Test Driver:

The test driver was a Pentium II @233 megahertz with 128megs of RAM and a 3Com 10/100 NIC, the controller was a Pentium @166megahertz with 128 megs of RAM & a 10mb NIC. Originally the testing software was a perl script, written in house, that copied ~750 megs of user-generated data to and from the server under test. This script was eventually followed by NetBench version 5.01, a ZiffDavis testing suite recognized throughout the industry as a test driver for PC benchmarking. To maintain test continuity we continued to run the perl scripted test as well as the NetBench, and both numbers are presented herein.

The test data used by the perl script was ~750 megabytes of user files, ranging in size from 3k to 137megabytes, consisting of 1537 files in 283 directories. The initial perl script just copied the files up from the test driver to the server, with no random writes, and no reads, other than those used to verify the writes. This was eventually replaced by NetBench v.5.01.

2.2 Network Configuration:

The testing network was made up of a Cisco 300 series 100mb FastHub, a Synoptic 10mb hub, a Cisco 5500 Switch with a 48 port 10mb card & a 24 port 10/100mb card. The box under test was always given it's own switched port at 100mb, while the test driver was connected to the 10mb hub, which was in turn connected to a 10mb port on the switch. The controller was connected to a separate 10mb hub, which in turn was connected to a 10mb switched port on the 5500. All the machines were on the same logical network & VLAN, thus minimizing impacts on the test derived from packets between networks.

Diagram 1

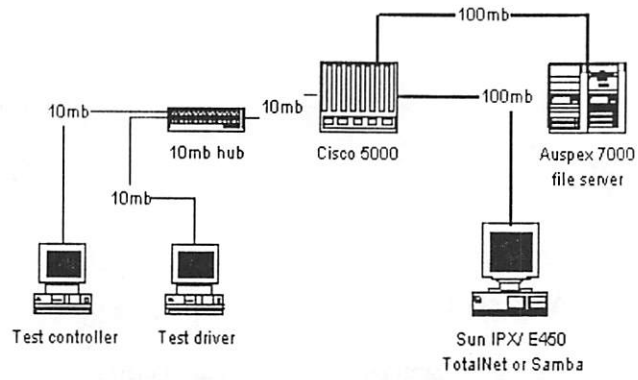
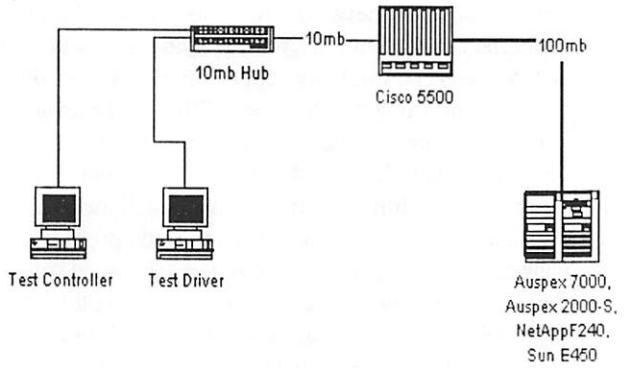


Diagram 2



2.3 Platforms Under Test:

2.3.1 TotalNet Advanced Server by Syntax:
Hardware configuration: Sun IPX, 128 MB RAM, quad 10/100mb Base-T NIC, Cycle5 processor upgrade
Software version: 5.2
See diagram 1

2.3.2 Samba:
Hardware configuration: Sun IPX, 128 MB RAM, quad 10/100mb Base-T NIC, Cycle5 processor upgrade
Software version: 1.9.18p10
See diagram 1

2.3.3 NetServices 1.0 beta by Auspex:
Hardware configuration: Auspex 7000/250, 512 MB RAM, 10/100mb Base-T NIC, 90Mhz HyperSPARC host processor
Software version: 1.0
See diagram 2

2.3.4 Network Appliance F230 server:
Hardware configuration: Network Appliance F230, 128 MB RAM, 4 MB NVRAM, 10/100mb Base-T NIC, Pentium II
Software version: ONTAP 5.1
See diagram 2

2.3.5 SunLink Server 1.0 beta by Sun:
Hardware configuration: Sun E450, 2 GB RAM, 4 x 300MHz CPUs, 10/100 Base-T NIC
Software version: 1.0
See diagram 2

2.3.6 SunLink Server 1.1 beta by Sun:
Hardware configuration: Sun E450, 2 GB RAM, 4 x 300MHz CPUs, 10/100 Base-T NIC
Software version: 1.1
See diagram 2

2.3.7 NetServices 2.0 beta by Auspex:
Hardware configuration: NS2000-P, 640 MB RAM,
Software version: 2.0
See diagram 2

2.3.8 Samba:
Hardware configuration: Sun E450, 2 GB RAM, 4 x 300MHz CPUs, 10/100 Base-T NIC
Software version: 1.9.18p10
See diagram 2

3. Test procedures:

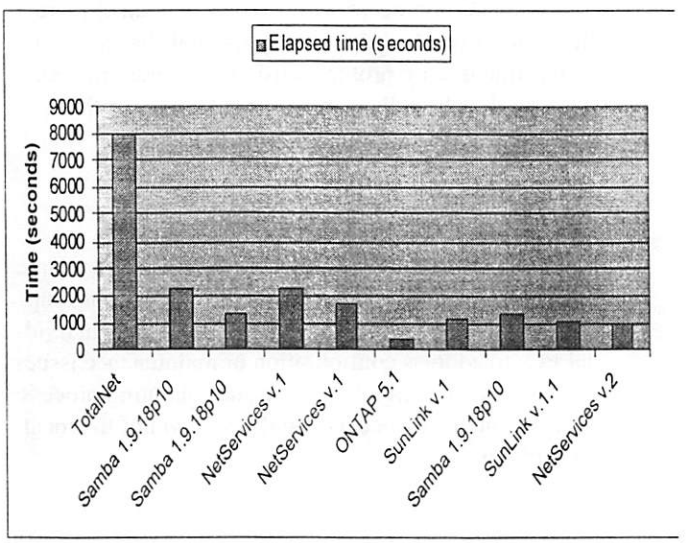
In all cases the host being tested was connected directly to a port on the 5500 switch, set to either 10mb or 100mb. A perl script was written that copied the test data to the server under test. Initially tests were run using a 10mb connection to the system under test, then unmounted and remounted using a 100mb connection when appropriate. Eventually we stopped doing the 10mb testing, under the assumption that the production system would be only connected to the switch via a 100mb port. Each test was run three times, each time during a different part of the workday, and the resulting time was averaged. In all cases the tests were run under working network load to reflect actual working conditions. The results are in elapsed seconds from start of write to completion of write.

4. Results:

The raw data results were:

Software tested	Elapsed time (seconds)
2.3.1 TotalNet 10mb	7980
2.3.2 Samba 1.9.18p10 10mb	2209
2.3.2 Samba 1.9.18p10 100mb	1385
2.3.3 NetServices v.1 10mb	2209
2.3.3 NetServices v.1 100mb	1694
2.3.4 ONTAP 5.1 100mb	426
2.3.5 SunLink v.1 100mb	1109
2.3.6 SunLink v.1.1 100mb	1078
2.3.7 NetServices v.2 100mb	910
2.3.8 Sambas 1.9.18p10 100mb	1291

A graph of the data is:



5. Sources of error

The potential sources of error introduced by our methods are many, ranging from individual server load variations to network broadcast storms or WINS elections. This is very deliberate, in that we wanted tests run in production networks on production servers while work was being conducted. Running each test multiple times throughout the work day should average out the larger impacts, yet still present a realistic spread of measured performance. By confining the network to our server room, and isolating most of the traffic within the backplane of the 5500 switch, a large amount of unrelated network traffic was avoided.

An additional source of error was the linearity of the original perl test script, which drastically favored the design of the Network Appliance architecture over any of the others in terms of raw performance numbers. An attempt was made to address this near the end of this project by using NetBench, but since all of the platforms had not been subjected to its test battery, the NetBench numbers are presented as additional information relative to the boxes tested with it.

6. Additional factors in the decision:

6.1 End user ease of use

For all of the solutions tested, end user impact was minimal. With no local client to install, configure, and maintain, all of our users have enjoyed an increase in stability, usability, and uptime. Additionally, the user community now only has one utility for accessing distributed resources rather than two or three, making their communication process with support personnel more clear and direct. Finally, the support staff has a much easier time tracing problems using only one protocol stack on the client PCs, rather than the two or three stacks that were previously installed.

6.2 Ease of administration

By centralizing our PC support structure onto a couple of servers we have largely eliminated the ongoing support costs relating to man hours spent visiting individual PCs to address configuration or maintenance issues. We have vastly simplified the troubleshooting process when problems do occur with the removal of the local NFS client.

6.3 Cost

As tested most of the configurations we explored ranged in price from $100,000.00 to $140,000.00 on the hardware solution side, including their related software cost. From a software solution side the prices ranged from free for the Samba & SunLink packages, to ~$12,4000.00 for the TotalNet package.

6.4 Support

At Tektronix we have an internal hardware support staff that is certified by our chosen vendors to do onsite maintenance and repair, with an average response time within a 2 hour time frame of problem notification. Computer Service and Support, or CSS, keeps a significant selection of spare parts and equipment on hand, but only for certain hardware configurations. One of the issues we had to take into account when evaluating hardware platforms was whether or not CSS was going to have a supply of spares on hand. In the case of both the Sun and Auspex hardware the spares were available.

6.5 Training

Our training costs should be reduced with the elimination of multiple client configurations and multiple client versions in concurrent use. Additionally, the training cost for troubleshooting should be reduced due to the simplification of client-server connections.

6.6 Scalability

In all cases the systems we evaluated should scale from the .5 terabytes currently in productions to the 1.5 - 2 terabyte data size we foresee in the next 18 - 24 months.

6.7 Ease of integration into our current environment

Given that CESA's job is primarily Unix server design and administration, the centralizing of PC support onto some sort of Unix server platform made the most sense in terms of ease of integration and training costs. This is mainly due to two factors: 1)the ability to leverage the experience base of the current administration staff and 2)the ability to leverage existing hardware for the initial phases of testing, and potentially early implementation as well.

7. Further avenues of work

We are continuing the testing process with an eye toward long term purchases for our server environment. A reality of our environment is that the data needs will exceed our current availability within the next six months, and a short term disk purchase will not address the long term scalability to the terabyte plus size we foresee in the next year or two. One of the primary goals is to re-write the current perl script to incorporate randomness into its write pattern, as well as random read structures. This change should then better reflect the across the board performance of all the platforms being tested, without favoring any particular design. Additionally, it will be written to run on multiple clients simultaneously, thus eliminating any discrepancies caused by some local load on the testing PCs.

8. Conclusion

All of the systems tested during this process would meet our needs as initially described, that of serving files between the PC and Unix environment without the need of a local NFS client on the PCs. One of the platforms tested showed a distinct advantage from a pure performance standpoint, but the largely proprietary nature of its architecture, as well as internal supportability issues relating to the hardware, kept it from being our choice. At the time of this writing the two choices we are going to pursue further are 1) Samba, running on existent hardware or on newer, larger hardware, or 2) Auspex NS2000-R. The rational for these choices includes familiarity with the hardware and software of the Auspex boxes, internal supportability by CSS of the Auspex or the Sun hardware, open source and support response time for the Samba package, and ease of implementation of either system.

Administering Windows NT Domains Using a non-Windows NT PDC

Gerald Carter
Engineering Network Services
Auburn University
jerry@eng.auburn.edu

Abstract

The chronicles kept by an explorer as he or she journeys into new territories can be invaluable for those who come later. The writing describe successes and failures, warning and hints to other possible solutions yet unfound. The evolution of a network is not different than the Lewis and Clark expedition of the early 1800's. Each day we learn something new. Either our proposed solution worked or it did not. It was either a dead-end or perhaps simply a detour.

In the Fall of 1997 with the release of Samba version 1.9.18alpha1, the College of Engineering at Auburn University began to experiment with the possibility of using a non-Windows NT machine (i.e. Samba running on a Solaris server) as a Primary Domain Controller for Windows NT desktop clients. This journey has continued through the present and is still progressing.

This paper will be a post-narrative of that journey. It is our belief that the administrative models and tools developed to support this environment will also be beneficial in other integrated environments, either in parts or as a whole.

This journey will be divided into in five topics: (1) relevant details regarding the configuration of the Samba PDC, (2) user account management, (3) remote file and printer access, (4) remote administration of clients and servers, and (5) the remote updates of systems, applications and associated settings. Each section will remain distinct enough so as to allow the reader to implement the varying portions of this paper independently.

Defining the Issue at Hand

The College of Engineering network is like many networks. We support a wide range of users each desiring different services. Some users require access to UNIX servers running various applications while others find one of the various Windows platforms to be their OS of choice.

When integrating heterogeneous networks, one possible solution is to maintain separate networks in parallel. For example, each network maintains the servers necessary for implementing services to support its associated clients even though each network may logically exist on the same wire. Services such as access to network printers are duplicated across server platforms. The parallelism is made transparent to the user by synchronizing the user account information.

The other possibility for integration is to centralize primary services such as user authentication on one server OS. The server must then support the functionality necessary to allow clients of other operating systems to become citizens of the network.

The College of Engineering at Auburn University has selected the second option. Our goal has been to provide continuity of services wherever possible without sacrificing the stability of these services. For example, users are able to access the same disk space as their home directory when logging into a PC or logging into a UNIX workstation, and network printers can also be accessed from either client platform. Perhaps the most important point in achieving this consistency is that each user is always authenticated with the same username / password pair.

Justifying the Trip

Regardless of where one is employed, any project that involves the expenditure of effort or resources requires some justification prior to approval. Therefore before continuing, I will provide the rational that began the exploration of using a non-Windows NT Domain Controller.

The first reason for entertaining this idea was the current existence of a stable Solaris based UNIX infrastructure. Global user authentication, as well as

ubitquous remote file access, were already implemented. If possible, we wished to leverage off existing services, both hardware and software, rather than replicate and reimplement them.

Another reason for the desire to keep primary services on one server platform was the shortage of support staff to maintain another operating system. The addition of another OS in the server room does not increase support costs linearly but rather exponentially. The reason for this can be accounted for in keeping up to date with platform patches, system tuning, and general knowledge of the workings of an operating system.

Implementation of a Samba Controlled Domain

Many of the services provided by the Engineering network are based upon open source software projects. For example, BIND and Sendmail are used for DNS and mail service respectively. Therefore, we were comfortable with the idea of providing some of the support for server packages ourselves. With this in mind, Samba (www.samba.org) was selected as the CIFS server over other packages such as the TotalNET Advanced Server.

Details of the implementation of the Windows NT Domain Control protocol can be found in [1] as well as in documentation included with the Samba source code distribution. Currently Samba does not "officially" support Domain Control for Windows NT clients.

Of course this is not the same as domain control for Windows 9x and older Windows clients. These clients use an entirely different mechanism for user validation and are not members of a domain in the real sense of the word.

The reason for the unofficial label is that not all of the functionality has been implemented. For a current list of implemented features and progress on remaining functionality, the reader is referred to [2].

At the time of the writing, one of the major items which has not been completed is the capability to participate as a domain controller in inter-domain trust relationships. For this reason, the College of Engineering chose to run a single, global domain for all Windows NT clients located within the College of Engineering. Currently the number of NT clients is around 50 with the majority of these being located in a public workstation lab. The number of users supported in this domain is approximately 1,700 with over half being regular users of NT clients.

It would have been just as easy to configure multiple, isolated domains. The reason for this that replicating Samba's account database is trivial when done via a secured `rdist` or more preferably by using `scp`. Future work in Samba will allow for other database backends, such as LDAP which provide for even easier replication, to be used.

Because this paper is more concerned with the model used in administering a non-Windows NT Server controlled domain, the reader is referred to [3] and [4] for details regarding configuring Samba in this capacity.

There is however, one detail that is pertinent to later discussions contained within this paper. This is the method used for storing user accounts within Samba. I have already alluded to this when discussing replicating Samba's list of accounts.

The SMB (a.k.a. *C*ommon *I*nternet *F*ile *S*ystem) protocol supports two levels of transmission of user credentials, or passwords (see [5]). The first is to transmit the password in plain text. In this mode, Samba is able to validate user connections against the standard UNIX password database (i.e. /etc/password or the network equivalent such as a NIS map).

The second method of validating user credentials is to use a challenge-response exchange. The details of this are explained in [6]. This mode requires the SMB server have previous knowledge of the user's hashed password. When SMB password encryption support is enabled in Samba, the account password hashes are stored in a file generally named `smbpasswd`. This is not to be confused with the tool used to manually manipulate these entries that is also named `smbpasswd`.

In order to support domain logons from an NT client, Samba must be configured to support password encryption. It also should be mentioned that supporting encrypted passwords is an "all or none" situation on a per connection basis. The reason for this is that the encryption capabilities of the server are negotiated during the session setup when in user level security (the reader is referred to [5] for a description of connection setups in the CIFS protocol).

Therefore, maintaining a Samba PDC also requires maintaining a separate list of NT accounts which is separate from the accounts listed in `/etc/passwd`. In fact, the only joining field between the two account lists is the UNIX uid which must be valid to control access to the underlying file system.

User Account Management

This brings up the topic of user management in a Samba controlled Windows NT domain and how to synchronize this information with the existing UNIX user accounts. The first issue to be addressed was how to initially populate Samba's `smbpasswd` file given that the College currently supported over 6,000 active user accounts. An active account is defined as one that has been accessed in the past six months.

In order to generate the necessary hashes, Samba will need access to the plain text of a user's password. In the general case, this is impossible and at the very best impractical due to UNIX's non-reversible password encryption algorithm.

There are two plausible means of performing this account population. The first is to use Samba's `update encrypted` parameter and require that users connect to a Samba server that does not have password encryption enabled. Once the `smbd` daemon successfully validates the user's plain text password, it will generate the respective password hashes and store them in the `smbpasswd` file. Once all passwords have been captured, a site can successfully migrate to encrypted passwords transparent to the users.

The second solution is to develop a custom application that will capture the user's plain text password and then generate the hashes itself. Of course, the most obvious chance to capture the plain text of a password is when the user is changing his or her password. This is the solution explored in this paper. **Figure 1** illustrates what occurs when a user changes their password from an Engineering UNIX workstation.

Note that the password is only validated against the UNIX account entry (i.e. NIS). If the change is successful, then the new password is written to the `smbpasswd` file with no questions asked.

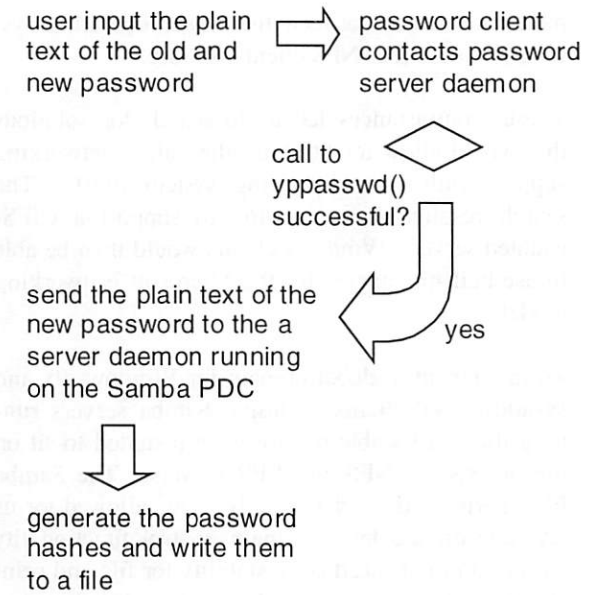


Figure 1: Flowchart of a user password change

By replacing the standard `/bin/passwd` with a custom in-house script, user accounts in the smbpasswd password file can be created using the same means for new and already existing accounts. This also provides a means to keep the password entry in both `/etc/passwd` and `smpasswd` synchronized.

For details on the availability of this tool, the reader is referred to [**10**]. Currently, the password change client is only available from UNIX hosts. However, plans are to include a Win32 binary that would allow Windows users to change their password directly from their desktop. This customized solution is more acceptable for us than attempting to support Windows' native method of changing passwords.

Remote File and Printer Access

For several years, PC's on the Engineering network gained access to remote file systems and printers via a locally installed NFS client. This was an acceptable solution at the time. However, there were several associated drawbacks.

The most prevalent one was the fact that NFS connectivity had no native support in DOS / Windows clients and therefore required an installation separate from the operating system itself. This proved to be problematic as systems were upgraded and incom-

patibilities arose between the newer operating systems and the older NFS client software.

These circumstances led us to search for solutions that would allow for utilizing the native networking support within the operating system itself. The search resulted in a decision to support a CIFS-enabled server. Windows clients would then be able to use built-in support for the Microsoft Networking model.

To implement a global domain for Windows 9x and Windows NT clients, multiple Samba servers running the latest stable release were installed to sit on top of existing NFS and LPR servers. The Samba PDC performed validation only. This allowed for us leverage off the latest domain control functionality with a minimal sacrifice of stability for file and print services. The `smbpasswd` file is then distributed to the servers using a secure method.

As can be seen in **Figure 2**, this allows for providing some of the consistency in logon environments mentioned previously. Hardware already was in place to support various disk configurations for home directories and network applications. Therefore, Samba servers initially acted as a gateway to these file systems. Over time, the Samba servers themselves are beginning to accumulate local disk space which is used for providing applications that only need be accessible from a PC such as the case with roaming user profiles.

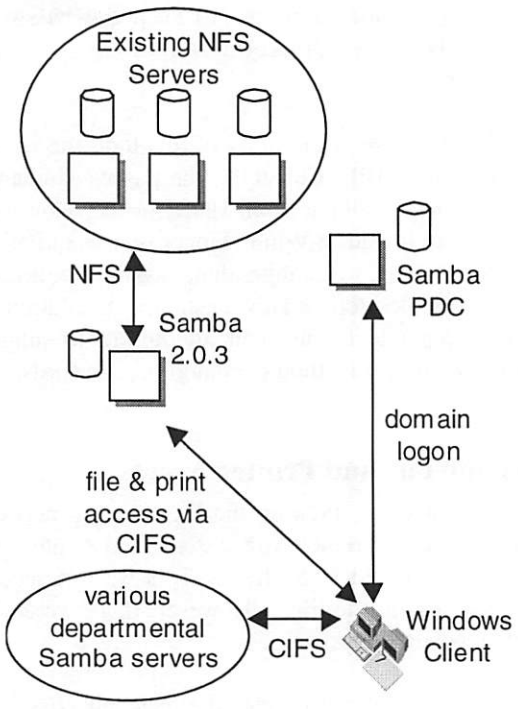


Figure 2: Overview of the College of Engineering network topology for PC's

The current system for our main Samba file server is a Sparc Ultra 170 with 384 Mb RAM running Solaris 2.6. The file server daily supports 400 - 500 Windows clients. Of these clients, normal use involves approximately 225 - 250 concurrent connections with each user having anywhere from 2 - 5 shares mounted. Statistics indicate that the load could comfortably increase to 300 connections with minimal detrimental effects given the current hardware configuration.

Network printers are accessed through the SMB server which then sends the spooled job to the remote printer via `lpr`. Rather than use the Solaris `lpsched` printing system, all servers are configured to use the Solaris port of the Berkeley `lpd` printing system.

Previously I mentioned that the Engineering Windows NT domain contained approximately 50 clients. This is true as this number reflects the actual Windows NT clients. However all PC's on the Engineering network utilize the file and printer services provided by various Samba servers. Domain control for Windows 9x and older Windows clients is provided by our main Samba box, which is currently running Samba 2.0.3.

Details of configuring Samba file and print servers can be found in the various documentation included with the Samba source code as well as various links from the main Samba web site (see http://samba.org).

Remote Administration of Clients and Servers

When administering a network, is it imperative to be able to automate common tasks such as process control, system reboots, and managing disk space. Maintaining a Samba controlled domain is, in many ways, identical to managing a true Windows NT domain. However, while some tools such as the Server Manager and User Manager for Domains allow access to information on the Samba PDC,

these applications are very inefficient for managing a large number of domain clients. When attempting to perform these duties on a mass scale, network administrators can turn to scriptable solutions such as Perl, Python, or one of the Windows based programming languages. The advantage of using a cross platform scripting solution is the ability to leverage off current programming knowledge and the reuse of existing scripts.

There are many possibilities for remotely accessing NT clients from another NT box. However, remotely administering NT clients from a UNIX terminal window takes a little bit of creativity. Through the use of free RSHD service for Windows NT (home.us.net/~silviu), UNIX network managers are able to perform basic NT administrative tasks without leaving their desktop.

The service does have some caveats that must be mentioned. The first is that the service provides no user authentication whatsoever. Therefore all commands executed by the rshd daemon will be done within the context of the account under which it is currently running. By default this would the LocalSystem account. However, we have reconfigured the service to run as a local administrative account.

It is a requirement that it can be guaranteed, and with a great degree of certainty, who can possess the ability to `rsh` into a NT client. This is done by creating `%SystemRoot%\rhosts` and editing it to contain a single entry referring to the network NIS+ master. The file is then locked down with the appropriate file permissions to disallow viewing or modification by any account other than a local administrator. **Figure 3** illustrates access control to these machines.

It is the belief that if anyone was able to gain unauthorized access to our NIS+ master, the NT clients will become small potatoes. While not entirely correct, it is simply another example of leveraging off the existing UNIX infrastructure, in this case security. Also we have modified the `rshd` source to perform forward and reverse DNS lookups so as to ensure that the machine from which the `rsh` command is arriving from is actually who it says it is.

With the addition of the capability to execute commands remotely, the quest then becomes to build up a sufficient toolbox of command line tools to be able to perform necessary tasks. Many possibilities have been outlined in [7].

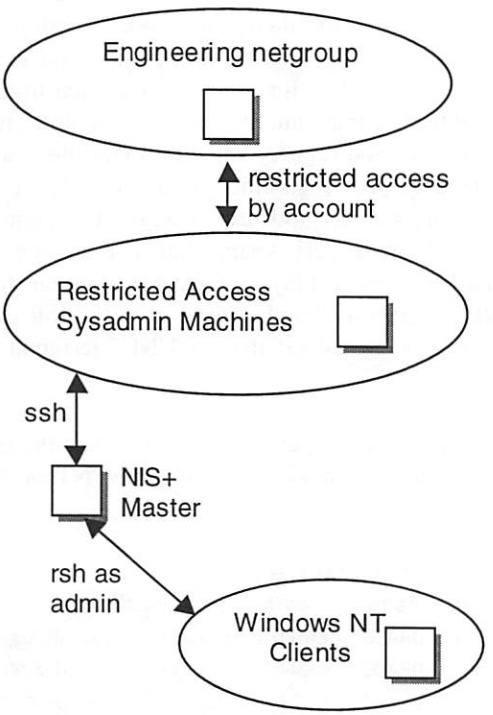


Figure 3: Restricting rsh access to Windows NT clients.

Here is an example of granting a Samba domain user named *toby* full permission to a share named *share1* on a Windows NT 4.0 domain member named *peeps*. The `sharegrant` utility is part of Pedestal Software's NT Security tools (www.pedestalsoftware.com).

```
$ rsh peeps 'sharegrant share1 toby:full'
Granting permissions to: ENG-NT\toby
```

Although the share ACL could be changed from another Windows NT machine, it cannot be guaranteed that one is always nearby. It is possible to always guarantee that telnet access to a UNIX workstation is available.

Perl has often been described as the kitchen sink of scripting languages and the Win32 port of the language is no exception. ActiveState's Win32 version of Perl5 (www.ActiveState.com) provides modules for managing NT user accounts, querying and

modifying registry values, retrieving information from the NT EventLog, as well as the traditional capabilities such as sockets and regular expressions.

The combination of an RSHD service and Perl provides a means for doing just about anything one could wish. To illustrate this point, I have made available several scripts that perform such things as configuring a machine in a public computing lab by setting common registry values. Again the reader is referred to [**10**] regarding the availability of these scripts as well as updates. **Listing 1** contains the output from a perl script that queries the local EventLog and displays information similar to the UNIX `last` command. Remember that all of this information is gained from a UNIX terminal window.

Listing 1: Using a perl script to determine the list of recently logged on users on a Windows NT workstation.

```
$ rsh beeps 'perl last.pl' | head –10
username machine  domain   logon - logoff
gaoyun1  DRACO    ENG-NT   May 31 10:46 1999 - still logged on
jingcai  DRACO    ENG-NT   May 30 17:51 - May 31 00:30 1999
yizhou   DRACO    ENG-NT   May 30 01:13  - May 30 01:18 1999
jingcai  DRACO    ENG-NT   May 29 18:56  - May 29 23:01 1999
millean  DRACO    ENG-NT   May 29 18:13  - May 29 18:40 1999
```

If we then wanted to find out which processes were currently running on beeps, we could use the `pulist.exe` command included with the Windows NT 4.0 Server Resource Kit [**8**]. Note that some of the output has been deleted for brevity.

Listing 2: Retrieving the list of running process from a Windows NT machine using the pulists.exe command.

```
$ rsh beeps '\\ivy\bin\ntreskit\pulist'

Process          PID     User
Idle             0
System           2
.....
rshd.exe         124     BEEPS\admin
NDDEAGNT.EXE     157
EXPLORER.EXE     130
systray.exe      59
F-AGENT.exe      191
OSA.EXE          188
xwin32.exe       190
CMD.EXE          194     BEEPS\admin
pulist.exe       88      BEEPS\admin
```

Then if we need to kill the X Windows Server process (pid 190), we could use the kill command, also include with the Windows NT 4.0, Server Resource Kit.

```
$ rsh beeps '\\ivy\bin\ntreskit\kill 190 '
process #190 killed
```

One final example of utilizing command line tools for monitoring purposes is given in **Appendix A**. This batch file is run on a Windows NT client at boot time via the AutoexNT service included in [**8**] . Each client has been configured to save the memory dump in the case of a system crash ("Blue Screen of Death"). The script, among other things, checks for the existence of the %SystemRoot%\memory.dmp file. If the file is located, an appropriate message is emailed to the network administrators indicating that the machine crashed. If the file does not exist, then it must have been power cycled.

Remote Updates of Systems, Applications and Associated Settings

Without the existence of a true Windows NT PDC, certain tools provided for remote updates of clients, such as Microsoft's System Management Server (SMS), become unavailable. The reason is that such tools often only run under Windows NT and must be installed on an NT PDC or BDC. Alternative solutions for automatically updating remote systems had to be developed. The solution was to utilize a method that had previously been developed for updating UNIX machines. The mechanism is fully explained in [**9**]. Since the publishing of the original Patch32 system, the method has been expanded to deploy applications such as anti-virus software, Netscape Communicator 4.08, and various network connectivity applications.

In addition to deploying updates and applications, the main Perl script was modified to email the logfile of any changes that were applied. In this way, system changes can be logged and monitored (see **Listing 3**).

Listing 3: E-mail message that acts as a log for monitor remote system updates.

```
Subject   : Machine Patched [WinNT] : mepc47
  Date    : Thu, 27 May 1999 17:07:06 -0500
  From    : admin@eng.auburn.edu
    To    : pcpatch@eng.auburn.edu

Installing hostex4...
 \\ivy\patch32\winnt\1381\hostex4\install.inf

Installing the_net...
 \\ivy\patch32\winnt\1381\the_net\install.inf
```

The details of deploying applications via the Patch32 script are beyond the scope of this paper. However, the point of interest are identical to deploying operating system updates and can be gleaned from the previously mentioned paper.

What Have we Learned?

The success with which we have been able to implement a Windows NT domain without a true Windows NT Server acts as proof of concept. It is possible to provide a Windows NT environment for users without great sacrifices to the services that a true Windows NT domain would offer. Of course there are differences, but from the user's point of view these are minimal. After all, it is the user we are attempting to best serve, is it not?

These results are obviously due primarily to the development of Samba and its associated functionality. There are, however, other issues that are not solved simply by the ability to support a domain logon. Other issues such as user management, remote administration and remote updates must be addressed as well. Just as all of these topics can not always be addressed solely by the Windows NT operating system, neither can Samba be the end all be all of an NT domain. Other tools do exist or can be developed with some effort to supply the necessary functionality to completely manage a Windows NT domain. Whether this is the best solution for your network, the reader is advised to consider all the costs and benefits.

Of the five topics covered, only two are tied to Samba and its implementation as a Primary Domain Controller. File and Printer service, remote administration, and remote updates of systems could all be implemented in a pure Windows NT environment using the techniques presented in here.

References and Resources

[1] Leighton, Luke Kenneth Casson, "NT 3.5/4.0 Domains for UNIX", Conference proceedings from the First Annual Large Installation System Administration of Windows NT, 1998.

[2] "Samba FAQ for NT Domain PDC Support", Available at all mirrors of the Samba web site (http://samba.org)

[3] Carter, G., "Linux Rules the Domain", Linuxworld, http://www.linuxworld.com/linuxworld/lw-1999-05/lw-05-thereandback.html, May 1999.

[4] Carter, Sharpe, Sams Teach Yourself Samba in 24 Hours, Sams Publishing, 1999.

[5] Carter, G., "How to cross the sometimes tenuous bridge between Linux and NT", LinuxWorld, http://www.linuxworld.com/linuxworld/lw-1998-10/lw-10-thereandback.html, October 1998.

[6] Visser, J., "On NT Password Security", Open Solution Providers, http://www.osp.nl/infobase/ntpass.html

[7] Carter, G. "Command-line NT: It does exist!", LinuxWorld, http://www.linuxworld.com/linuxworld/lw-1999-04/lw-04-thereandback.html, April 1999.

[8] Microsoft Windows NT Server Resource Kit, Microsoft Press, 1996.

[9] Carter, G., "Patch32 : An System for Automated Client OS Updates", Conference proceedings from the First Annual Large Installation System Administration of Windows NT, 1998.

[10] Web page for this paper, http://www.eng.aubun.edu/users/cartegw/non-NT_PDC

Appendix

%SystemRoot%\Systemr32\Autoexnt.bat script

```
rem ******************************************************
rem ** Environment variables                             **
set LOGFILE=%SYSTEMROOT%\local\log\autoexnt.log
set LOCAL=%SYSTEMROOT%\local
set ADMINMAIL=cartegw@eng.auburn.edu
set SUBJECT=host %COMPUTERNAME% rebooted

rem ** Timestamp the log file **
echo :::::::::::::::::: > %LOCAL%\temp.log
date /t >> %LOCAL%\temp.log
time /t >> %LOCAL%\temp.log
echo :::::::::::::::::: >> %LOCAL%\temp.log

rem ****** Start the Workstation service ******
%SYSTEMROOT%\system32\net start LanManWorkstation >> %LOGFILE%

rem ****** Update the system clock ******
%SYSTEMROOT%\system32\net time \\ivy /set /yes >> %LOGFILE%

rem ****** Apply any necessary patches ******
if not exist %SYSTEMROOT%\memory.dmp goto patch_os

   if exist %SYSTEMROOT%\local\memory.dmp del %SYSTEMROOT%\local\memory.dmp
   move %SYSTEMROOT%\memory.dmp %SYSTEMROOT%\local\etc\memory.dmp
   echo [%COMPUTERNAME%] : Blue screen! >> %LOCAL%\temp.log
   set SUBJECT=[%COMPUTERNAME%] : Blue screen!
   echo. >> %LOCAL%\temp.log

:patch_os
   \\ivy\perl5\bin\perl \\ivy\patch32\patch32.pl >> %LOCAL%\temp.log
   type %LOCAL%\temp.log >> %LOGFILE%
   echo. >> %LOCAL%\temp.log

type %LOCAL%\temp.log | \\ivy\bin\blat\blat - -t %ADMINMAIL% -f
     roundup@eng.auburn.edu -s "%SUBJECT%" -server mailhost.eng.auburn.edu
del %LOCAL%\temp.log

rem ****** set the chkdsk flag to check the hard disk on reboot
echo y| chkdsk c: /f
```

Radio Dial-in Connectivity to NT Networks

Kenneth O. May, P. Eng
ISM, a Member of IBM Global Services

1 Preface

"Global-village is the world viewed as a community in which distance and isolation have been dramatically reduced by electronic media"; *Item #307 (11 Dec 1998 05:00) - global village: M-W's Word of the Day: Web:* http://listserv.webster.m-w.com/SCRIPTS/WA-MERRIAM.EXE?A2=ind9812B&L=MW-WOD&P=R242

This paper discusses a limited case study of a specialized use of Windows NT dial-in connectivity to create a global village. Namely a radio link to a telco dial-in connection with a remote modem for network connectivity.

2 Abstract

Figure 1 - Travel on muskeg to Behan Compressor Station

2.1 Requirements for Network Access

It's 9 am on a cold January morning. George Walls, a TransCanada measurement tech, is sitting in the Behan Compressor Station waiting for the sun to come up. He just arrived after a 40 minute helicopter ride over the muskeg from the nearest road. Once the sun is up he'll calibrate the meter run at the station, but that's still several hours away since there's only 3 hours of daylight today.
He starts his PC and waits until the screen displays "User Name Verified". Now he can read his e-mail, do his expenses, submit his overtime and generally keep himself preoccupied while he waits in the dark. For George, the availability of the NT network is critical to his productivity, personal mental state and the company's productivity.

2.2 Site Locations and Communications Methodologies

The sites are generally located in the top half of Alberta, Canada and isolated by the lack of any communications methods other than a radio link to the nearest telephone connectivity point. Because of topology and costs, microwave is not a viable option and it's normally many miles to the nearest landline connection. Some of these connectivity points are telephone company sites and others are shared by companies who have similar communications problems. See Appendix A for a high level block diagram of the communications flow.

2.3 Limitations

Being a data radio link, the speed is restricted to 19200 baud. Since it is a dial-in the receiving modem has a 20-minute inactivity monitor and will drop the line after this time. The main limitation to connectivity is the time waiting for things to happen because of the relatively slow speed running Microsoft Outlook, SAP, Netscape etc.
As a security policy, the company has restricted dial-in access to their internal web only.
Any upgrades to the communications hardware or firmware have to be co-ordinated with the computing support areas to ensure that the system integrity is maintained.

2.4 Reliability Requirements

Data transferred over this link can affect billing to TCPL's customers. Problems of this type set the priority of the access to high, which is a 2-day response time for these sites. Generally, most of the problems are not of this nature, so that problem response time is from 2 weeks to a month for non-business-critical problems. Reliability of the communications at these sites varies. Some sites require a maintenance visit as little as once a year, while others tend to have up to 8 visits a year. Differences in weather conditions such as high wind areas, power stability and grounding conditions are key factors that affect this reliability.

3 System Operating Parameters

3.1 NT Configurations

The operating system is NT4.0 with Service Pack 2. RAS setup has UnimodemEnable=0. This is added to REGEDT32.EXE so that it enables the MODEM.INF file in C:\WINNT\SYSTEM32\RAS directory. This file has to be edited first for the modem type we used, In the same directory, SERIAL.INI must be blank with a single carriage return and RASPHONE.PBK will be edited from the Dial-up Networking Icon in My Computer.

3.2 Hardware configurations

A range of manufacturers hardware was used, although primarily Compaq or IBM Pentium 133 MHz or 166 MHz desktop PCs are used. The connection is direct connect with a serial cable to the data radio interface. This connection had to have pins 7 to 8 tied together so that RAS would recognize the link since the modem was at the other end of the radio link.

3.3 Software Configurations

Attached as Appendix B is a set of the instructions we use for the setup procedures for this environment. The target user for these instructions is an IS type person familiar with our NT and normal dial-in configuration, so some of the steps are combined and some actions are implied. We didn't want to make this into a 3 ring 2 inch binder.

The gist of the work is to ensure that all references to modem operating speeds are set to 19200 baud. This includes MODEM.INF and RASPHONE.PBK. Additionally, the Control Panel/Modem setting has to be set with speed of 19200 baud, Error Control and Flow Control OFF in the Connection/Advanced area of the modem properties. The reason we do this is to preclude the software sending any parameter changes to the modem as the parameters set by the Automation technicians in the modems at the other end of the radio link are critical to reliable operation of the radio interface.

It can also be noted that the modem properties used are for the AT&T Comsphere3810 Plus. Although other types of AT&T modems are actually used at various locations, the 3810 Plus initialization parameters worked more consistently than any of the others.

3.4 Network Configuration

The Network connection end of the dial-in is a modem bank connected to a Cisco router. Authentication for access is twofold. First, access to the network is validated with a POP3 authentication and then the NT name and password are authenticated for the domain. This double authentication seems to provide reasonable security for dial-in as the only address the can be traced on the network is the answering end of the modem bank. (Note this is the company standard dial-in service)

4 Installation Problems and Solutions

4.1 Problems noted during dial-in

4.1.1 Special Wiring requirements

The 9-pin connector on the COM port 9 pin connection must have pins 7 and 8 cross-connected to allow the RTS/CTS control to work properly on the PC. Please see Appendix C for the signalling configuration of 9 pin to 24 pin modem connections.

4.1.2 Registry Change

The only change is to disable the Unimodem default configuration for RAS, which enables MODEM.INF as the location for modem configurations based on the type selected. (see Appendix B for change details)

4.1.3 RAS file changes

Before the above changes are done, and Service Pack 2 is reloaded , the C:\WINNT\SYSTEM32\RAZ\SERIAL.INI file must be checked to ensure that it is empty. Another critical file which should be checked is C:\WINNT\SYSTEM32\RAS\MODEM.INF for the specified modem to ensure that the initialization string in use does not reset the modem parameters. These had to be custom configured at the modem site at the end of the radio link for the telco interface at that site. This was done in our case study by Trans Canada 's Automation technicians who are trained in this environment.

4.2 Order of work critical

Appendix A is the definitive list of tasks associated with setting up our Comsphere modem for the Radio Link dial-in sites. The order of the tasks is critical, for example if one changes the Modem.inf file, RAS needs to be reloaded and Service Pack 2 rerun.

4.3 File References

C:\winnt\system32\ras\modem.inf
C:\winnt\system32\ras\serial.ini
C:\winnt\system32\ras\rasphone.pbk
C:\winnt\system32\ras\switch.inf
Registry via c:\winnt\system32\Regedt32.exe

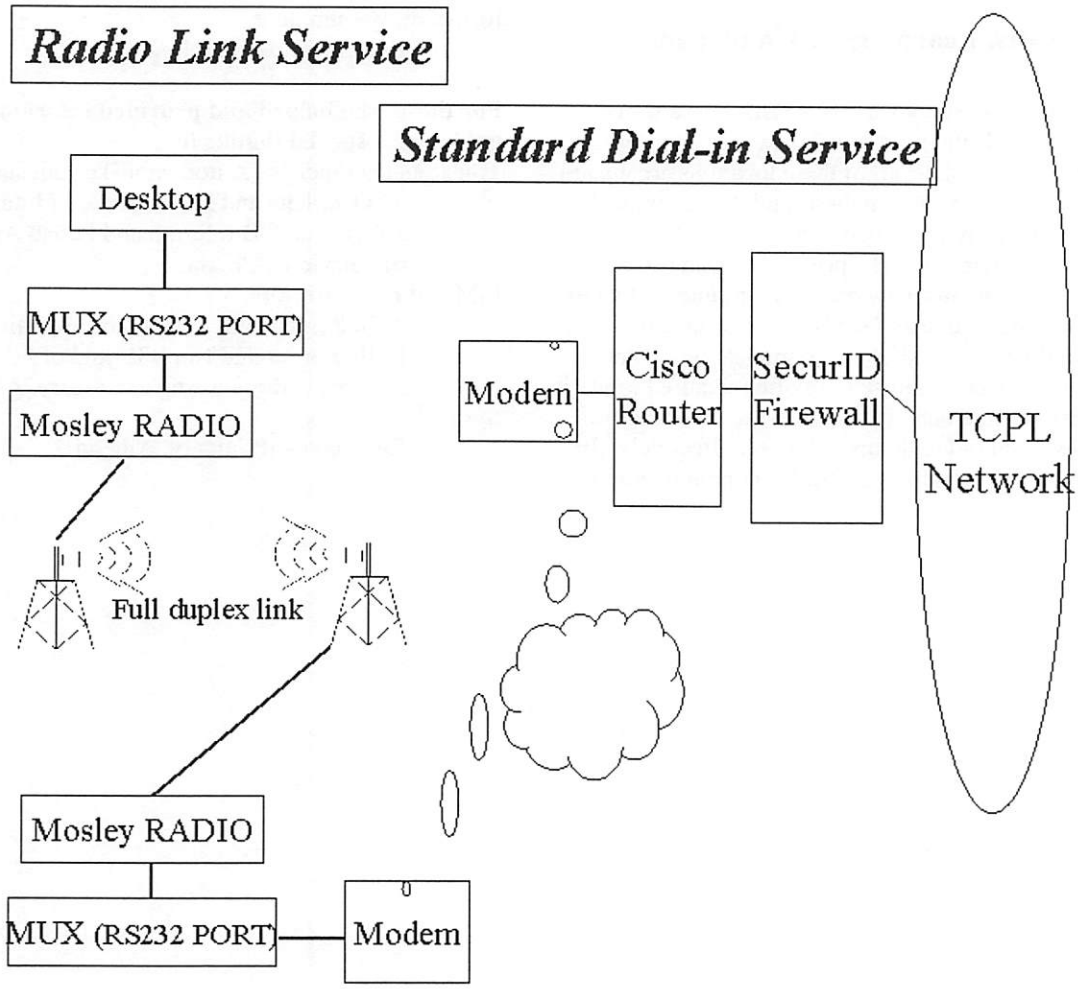
5 Results, Learnings & Challenges

This limited case study was not a test. These were all real operational situations which had to work. The result was that, to date, all of these locations are up and operational on a day-to-day basis and they provide the network connectivity needed to meet Trans Canada's business requirements. The primary learning from this experience is that in a networked environment, it takes a range of knowledge and skills to establish and maintain the integrity of the communications. Many people with diverse skill sets are often required to work together to ensure that even the most remote user has the access required to do his or her job effectively. In our case, it was Windows NT skills, communications integration skills, radio connectivity skills, and most important, people skills to bring it all together.
The challenges are to maintain the integrity of this connectivity as software and hardware upgrades are implemented. Such as the implementation of WindowsNT Service Pack 4. An ongoing challenge is to understand and, if possible, control the exact signal expectations of WindowsNT RAS service as it passes in and out of the COM port to the modem.

Respectfully submitted,
Kenneth O. May, P Eng

For those who helped and provided information and guidance, a special thanks to:
TransCanada Pipelines Automation Technicians
Al Ouelllette and Kevin Strate of Edmonton, Alberta, and Bill Smith and Pascal Andriano of Fairview, Alberta.
ISM and its contractors
Aain Ramcharan, Jana Kolot, Henming Lee, Kelly Fortier and Norm Bujold of Edmonton, Alberta, and Soon Ang of Calgary, Alberta.
SystemHouse Ltd.
Ray Peters of Calgary, Alberta

Appendix A


Radio Link Service
Standard Dial-in Service
Desktop
MUX (RS232 PORT)
Mosley RADIO
Full duplex link
Mosley RADIO
MUX (RS232 PORT)
Modem
Modem
Cisco Router
SecurID Firewall
TCPL Network

Appendix B

DIAL-UP Networking Configuration Setup for Radio Linked Sites.

Follow the RAS setup steps outlined on page 2-41 Titled Installing RAS (for Telecommuters).
* Note: IF RAS detects a **Standard Modem** - just continue. The modem will be manually changed later

The following procedure will replace the steps from Step 13 and onward.

At step 13 Restart the PC/ LAPTOP.
*Note: RAS needs to be configured to use the Modem.inf instead of the Unimodem.
*Warning: BE CAREFUL in editing the following Registry Entries.

1. Click on **Start**
2. Goto **Run,** type in **Regedt32** in the command box and press Enter.
   - Select **Hkey_Local_Machine**
   - Click on **Software**
   - Click on **Microsoft**
   - Click on **RAS**
   - Click on **Protocols**
   - From the Menu bar select **EDIT** and then **ADD VALUE**
   - In the Value Name box type in **EnableUnimodem**
     - **NOTE: The above IS case sensitive with NO spaces.**
   - In the Data type box change to **REG_DWORD**
   - In the **DWORD** editor window type in **0**(zero) as the data
   - Click **OK**
   - Close **Regedt32**
3. Goto **Control Panel** and click on the **MODEMS** icon
4. Remove the Modem that was found by RAS at setup.
5. If there is a prompt to update RAS select **YES**
6. Now **RESTART** the computer.
7. If there is a prompt to detect the modem go **CANCEL**
8. Once the computer is back up **Log on** and double click on **Dial-Up Networking**
9. This will cause **RAS** to Prompt for a Modem
10. When asked to auto detect the Modem click **NO**
11. From the Configure Port Window select the **AT&T Comsphere 3810 PLUS 288**
12. Under the **Port Usage** select the **Dial out** and **Receive** calls option
13. Click **OK**
14. Click **Continue**
15. At the RAS Server TCP/IP Configuration window click **OK**
16. At the RAS Server IPX Configuration window click **OK**
17. At the Windows NT Setup window type in the path to the I386 folder. Typically **C:\NTDRIVER\I386**
18. From the RIP for NWLink IPX Configuration window select **NO**
19. Close any windows and **RESTART** the computer (Note: The Event Viewer will display the following error when Windows NT restarts. "The RIP for NWLink IPX service failed to start due to the following error: The parameter is incorrect.") Ignore this. Reinstalling SP2 will solve the problem.
20. Once the computer is back up **Log on** and Reapply Service Pack 2
21. In the **NTDRIVER** folder open the **SP2/BATCH** folder and double click on **SP2LAP.BAT** for both Desktops and Laptops. This is because the **SP2LAP.BAT** includes updates to the RAS system which are not included in the **SP2DESK.BAT.**
22. When this is done the computer will prompt you to **REBOOT** select **YES**
23. After **SP2** is reinstalled check for the following:
    - Set the modem speed to 19200 bps in the following places
      - Go to Control Panel - Modem.
      - Under the General Folder, make sure that you have the correct modem installed and NOT Standard Modem. If you still have Standard Modem - Remove and Add back the correct modem.
      - Click on Properties and Set the Maximum speed to 19200.
      - Click on <OK>.

- GO TO My Computer then Double Click Dial-Up Networking.
- Select the appropriate phone entry (ex. Long Distance NGTL without '9') If the phone book entries aren't standard, add an updated copy of RASPHONE.PBK.
- Click on <More>
- Click on <Edit entry and modem properties>
- Click on <Configure> beside the Dialin using line.
- Set Initial speed (bps) to 19200
- Click on <OK>

23. Set the Frame Type of the NWLink IPX/SPX Compatible transport to Ethernet 802.3

    - Go to Control Panel - Network
    - Click on the Protocols Folder
    - Click on NWLink IPX/SPX Compatible Transport
    - Click on Properties
    - Change Frame Type to Ethernet 802.3
    - Click on <Apply>
    - Click on <OK>

24. Make sure that the Enable IP Forwarding check box under the Routing Folder of the Microsoft TCP/IP Properties is NOT Checked.

    - Go to Control Panel - Network
    - Click on the Protocols Folder
    - Click on TCP/IP Protocol
    - Click on Properties
    - Click on the Routing folder
    - The box beside Enable IP Forwarding should NOT be checked.
    - Click on <Apply>
    - Click on <OK>

25. At the Network Settings Change window, click on <YES> to restart the computer.

Check in C:WINNT\SYSTEM32\RAS, that SERIAL.INI in blank, and the SWITCH.INF file is for Calgary Dial-in Server. If not copy these files to this directory before starting the dial-in function.

After the PC has restarted continue with pages 2-42 and 2-43 to complete the setup.

NOTE:

1. Although all of this can be done in advance at a LAN site, you may find that it needs to be redone when your on-site. Generally, one would start over by removing the RAS COM-port and adding it in. Then continue with the rest of the install.
2. If the PC is connected into the building wiring which only has 6 wires, the 9-pin connector on the COM port must have pins 7 and 8 cross-connected to allow the RTS/CTS control to work properly on the PC.

Appendix C

9 -Pin Cabling for Mosley Radio sites.

The following table shows how to connect a 9-pin serial port connection on a computer to a 25-pin connector on a modem. Again, if you are using an off-the-shelf cable, be sure all the pins are connected as shown in the table.

Note that some modems have the Data Set Ready (DSR) signal physically tied to the Data Carrier Detect (DCD) signal. Some 1220-bps modems and other 2400-bps modems have dip switches default to this setting as well. As a result. If such a modem loses power while listening for a call, the Remote Access server cannot detect the condition because the DSR will not change as it does with other modems.

9-pin serial port	25-pin modem connection	Signal	Notes
1	8	Carrier Detect	
2	3	Receive Data	
3	2	Transmit Data	
4	20	Data Terminal Ready	
5	7	Signal Ground	
6	6	Data Set Ready	
7	4	Request to Send	Mosley tie 7-8 together on 9-pin
8	5	Clear To Send	Mosley tie 7-8 together on 9-pin
9	22	Ring Indicator	Optional

Figure 1 - 9 pin to 25 pin chart

NT Security in an Open Academic Environment

Gregg Daly, Gary Buhrmaster, Matthew Campbell, Andrea Chan, Robert Cowles, Ernest Denys, Patrick Hancox, Bill Johnson, David Leung, Jeff Lwin
Stanford Linear Accelerator Center

Abstract

Stanford Linear Accelerator Center (SLAC) was faced with the need to secure its PeopleSoft/Oracle business system in an academic environment which only has a minimal firewall. To provide protected access to the database servers for NT-based users all over the site while not hindering the lab's open connectivity with the Internet, we implemented a pseudo three-tier architecture for PeopleSoft with Windows Terminal Server and Citrix MetaFrame technology. The client application and Oracle database were placed behind a firewall, and access was granted via an encrypted link to a thin client. Authentication in the future will be through two-factor token cards. NT workstations in the business system unit were further secured through switched network ports and an automated installation process that included SMB signing and disabling LM Authentication in favor of NTLMv2. The hardened workstations then accessed the business system through the Citrix Secure ICA client. How these security measures affected our mixed environment (Windows9x, Samba, Transarc AFS clients, Pathworks, developers, researchers) is discussed.

1. Security in an Open Environment

SLAC has 3800 hosts running Windows NT (1400), UNIX (2000), Macintosh (350) and several other operating systems. The network is based on a highly redundant Gigabit Ethernet backbone implementing virtual LANs allowing hosts on the same subnet to be located anywhere in the Lab. All external network traffic to the Internet passes through a single filtering router.

Like most academic research environments, SLAC does not have a real external firewall. The High Energy Physics (HEP) community has a tradition of being very open – the strength of such an environment produced the World Wide Web. (CERN, where the World Wide Web originated, is another High Energy Physics facility, while SLAC itself was the first web site in the U.S.A.) Current HEP experiments can have more than a thousand experimental collaborators (distributed internationally), and collect hundreds of terabytes of data per year. The very size of the collaborations involving software development, hardware design, implementation, and data analysis require incredible amounts of open discussion, collaboration, and gigabit network bandwidth with as few barriers as possible. Physicists are creative people who tend to enjoy playing with the latest software releases (beta) and in solving puzzles (how do I get around this restriction?). Once these users have found a way around restrictions or become used to certain software "features," it can be extremely difficult and politically inadvisable to (re)impose barriers that might delay large projects. However, business application systems used within the Lab, and the Windows operating systems they run on (even Windows NT), are not designed for use in an open Internet environment.

In mid-1998 there was a major security incident at SLAC. More than 25 machines had root compromises and the intruders used more than 50 user accounts. In order to assess the damage and clean up the systems, SLAC dropped off the Internet with respect to interactive services for a one week period (incoming web access and bi-directional e-mail were still allowed). Needless to say, this incident caused a lot of attention to be focused on computer security issues. An example of a change in attitude was that the NetBIOS over TCP ports were finally blocked at the SLAC external router with only a grumble of protest.

1.1. Constraints

To protect the business systems, we needed to implement credible responses to what seem to be the major vulnerabilities: compromised passwords; and the combination of "scientist maintained" workstations and a mode of thinking that "PC" meant "*personal* computer". We had to balance the need for a secure, reliable computing and network infrastructure with the need for an open collaborative research environment. The bottom line at SLAC is, "The Physics must get done!"

1.2. Threat Analysis

SLAC's business data is stored on an Oracle database server. Over 200 users access the data via PeopleSoft applications running on NT workstations from both onsite and remote locations.

In terms of possible threats to the business systems, the major point of vulnerability was considered to be the Oracle database machine. The attacks we wanted to protect against were external network-based attacks or attacks involving unauthorized but authenticated users (compromised passwords, etc.). Additionally, we needed an architecture adaptable enough to respond to new threats over the next two years. In particular, it should include elements making it resistant to keyboard monitoring programs like Back Orifice and Netbus.

1.3. Corporate-world Solution

As we started doing our research, it was clear the "standard corporate model" was to build a fortress. Corporate networks were behind firewalls which allowed almost no protocols to travel through them and often used proxy servers for people inside the firewall to have access to Internet services. There might be a sacrificial web or e-mail server sitting outside the firewall, but that was all.

This was not a viable solution in an academic or research environment. An important factor in a research environment is to provide for the unexpected. Would the web have been developed without that kind of open environment?

1.4. Mini-corporate Solution

Another possibility was to treat the business services people as sort of a mini-corporation within SLAC, and place them behind the kinds of barriers described above (Fig. 1). While that thought made the physicists happy for a few minutes, only a little reflection was necessary to realize the business services people *support* the rest of the lab. To perform their mission, they have to communicate with the rest of the Lab concerning budgets, personnel, and financial issues. The senior research

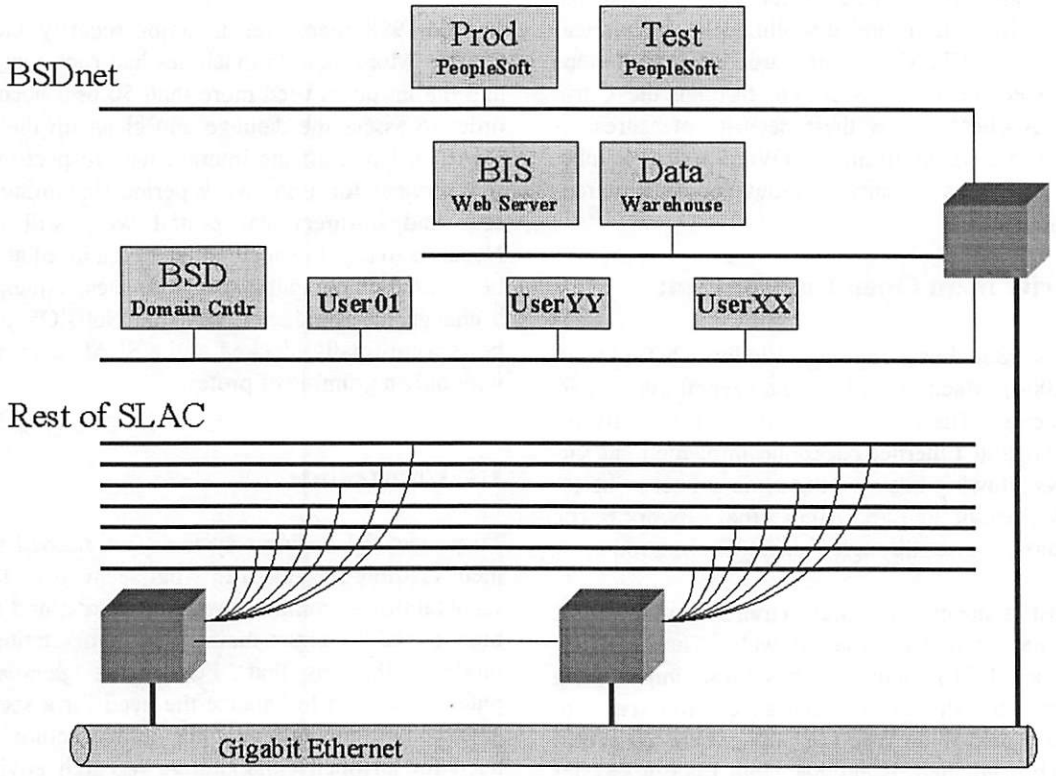


Fig. 1 Mini-corporate Solution

leaders are often heavily involved in financial aspects of the experiments and need access to current budget information, particularly near the end of fiscal cycles.

Another variation of this approach, with each business service user having two NT workstations – one secured and one configured for SLAC inter-operability – was also rejected. Besides the cost of additional workstations and networking equipment, the fear was that in the long run, users would bypass the security using either floppy disks, "yellow cables" or new ways of "piped" cross authentication (e.g., new web browsers) between the secure and not-secured systems.

1.5. Layered Solution

A layered solution finally survived a number of discussions and presentations (Fig.2).

1. Only the Windows Terminal Server/MetaFrame farm configured with PeopleSoft apps may access the database server (no direct workstation access).
2. Both the Oracle database server and this Windows Terminal Server/MetaFrame farm would be pulled into a subnet (BSD Extremely Private Network (EPN)) with very restrictive port filtering rules. Under normal operations, only these few machines would be behind this "firewall", and no personal workstations.
3. People directly using the business systems (as opposed to just viewing information through a web interface) would be required to access the PeopleSoft applications and the database server only via the MetaFrame 128-bit encrypted thin client.
4. Token cards provide two-factor authentication to this system.
5. Access to the MetaFrame servers must originate in the BSD subnet.
6. People directly using the business systems would also be required to have a standardized and security-hardened software configuration with basically interchangeable but fully functional workstations.
7. A few mission-critical users would have workstations limited to running only the business applications and not much else, and would be on the same BSD-EPN network as the servers.
8. Network and host-based Intrusion Detection Systems will watch network traffic and operations on the database server.

A major feature of this design is that during normal operations, the business users have full access to SLAC NT network resources. In times of response to an intrusion, two levels of "air gap" can be implemented to separate BSD-EPN and BSDnet from the rest of SLAC and the Internet, as shown by Fig. 3:

1. Mission critical functions can proceed with the few workstations on the same BSD-EPN network as the servers.
2. The 200 business users in BSDnet can be reconnected back to the servers on a priority basis after being revalidated. This subnet has its own domain controllers, SMS and home directory servers so it can exist as a standalone domain.

Selling the plan was simplified by two factors: the scientific researchers see no changes in their environment

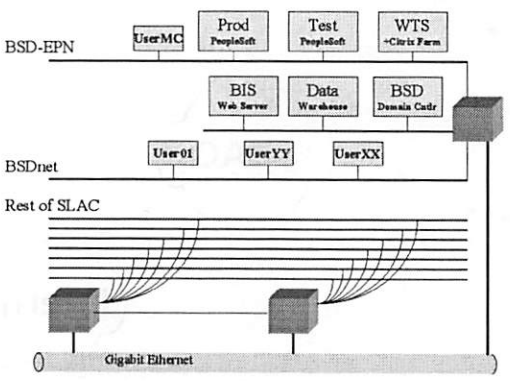


Fig. 2 Layered Solution

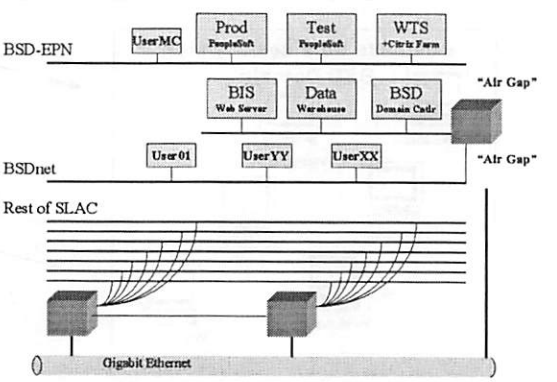


Fig. 3 Layered Solution during Intrusion

in the way of additional restrictions; and the "air gap" concept is easy for management to understand.

2. Configuring and Securing Windows Terminal and MetaFrame Server Farm

2.1 WTS/MetaFrame Overview

Microsoft's Windows NT 4.0 Server, Terminal Server Edition (commonly called WTS) is an extension to Windows NT Server which provides, in essence, a multi-user Windows system. That is, WTS allows multiple individuals "login" access to one Windows NT desktop, and attempts to keep each user separate from each other.

Citrix's MetaFrame product extends WTS in a number of areas. The most important features to SLAC were

- load balanced server farm
- Secure ICA client
- user drive remapping
- seamless windowing

The load balancing feature of MetaFrame allows administrators to publish an application, which will connect to the "least loaded" MetaFrame servers that supports that application – providing both load balancing and server failure recovery. (See Fig. 4) An added bonus for system administration is that a server can be placed "offline" for maintenance.

The Secure ICA client provides 128-bit encryption on the data stream. With password and session stealing the biggest threat to corporate networks, the ability to encrypt the entire session moves the complexity up one more level for a potential attacker.

Seamless windows allow an application to appear on the users' desktop as if the program was running natively.

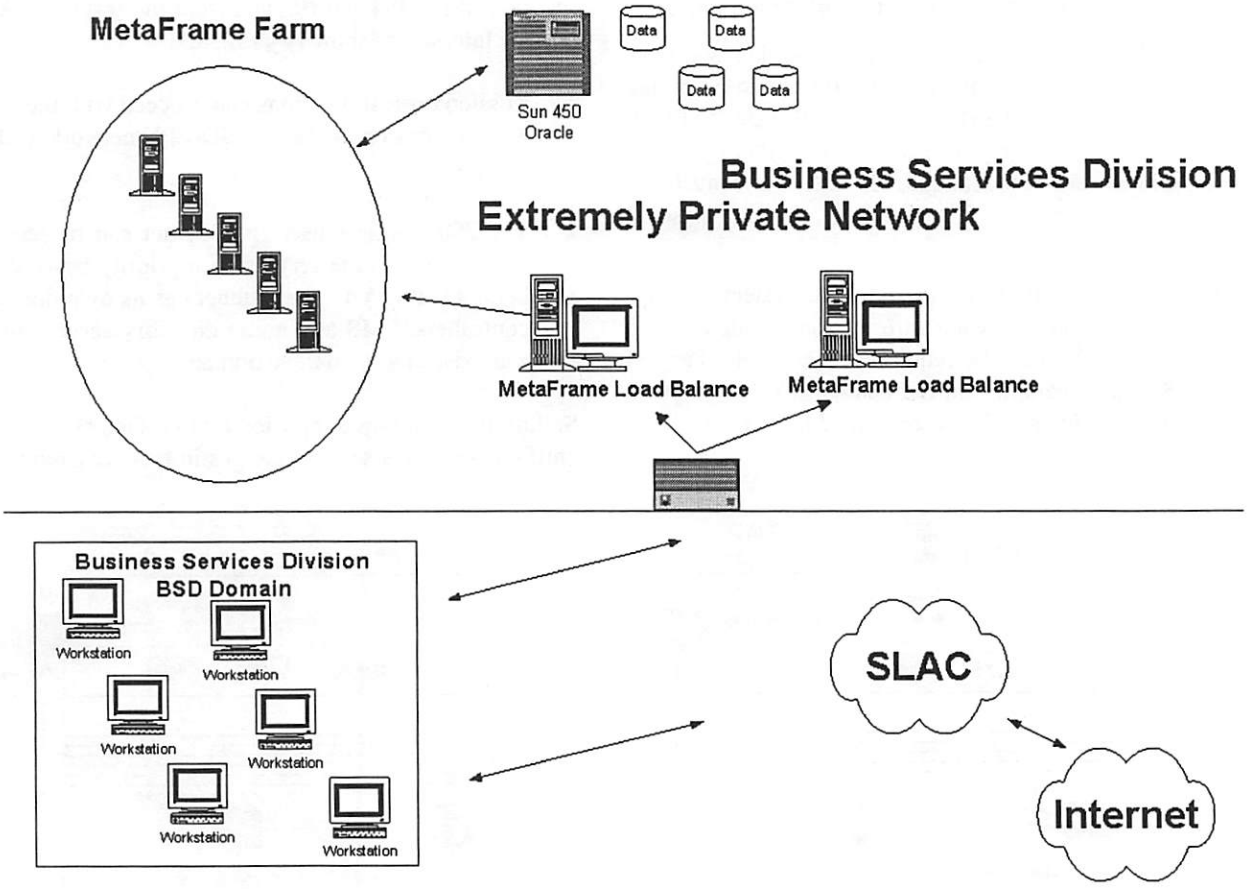


Fig. 4 PeopleSoft WTS/MetaFrame Farm

Along with user drive remapping through the ICA client, the user's application has access to the local PC's resources when that is necessary.

2.2 Configuring Applications

Microsoft's NT solutions tend to be designed with the assumption that one and only one person will have a desktop on the machine at once. At logon, a number of things happen behind the scenes to "customize" the particular machine one is logging into. A system which allows multiple logins at once needs to isolate these "customizations" to particular users. This requirement has led to some challenges given the underlying architecture of Windows since the system and applications were designed to assume that only one user was logged on at once. Many (common) system directories contain configuration files for the currently logged on user. The NT registry, one of the central repositories of customization data, consists of both user and machine specific sections. While some newer applications are "WTS aware," or at least "WTS friendly," older ones can be complex to correctly configure.

PeopleSoft is a vendor of complete Enterprise Resource Planning solutions in the Human Resources, Financials, Manufacturing, and Student Administration System arena. SLAC is currently running PeopleSoft 6 HR and FS. PeopleSoft 6 is a classic 2-tier application, where the business logic runs on the client system communicating to a DBMS server. In addition, PeopleSoft comes packaged with a set of 3rd party applications such as Crystal Reports (a report writer) and SQR (a report and batch processing program). PeopleSoft has its own development tools and runtime processing environment. SLAC uses Oracle as the backend RDBMS for PeopleSoft.

As a classic 2-tier client/server system, PeopleSoft provides no protection of the data flowing across the network, nor does it in any way verify that the application connecting to the RDBMS is actually PeopleSoft, and will follow the appropriate business logic. The passwords in the PeopleSoft database are easily decrypted, which would allow someone to login to the application as a user with full authority on the entire database. PeopleSoft admits that this is a weak link in security and has addressed this in later versions of their product.

PeopleSoft, Crystal Reports, and SQR were designed and implemented long before WTS was introduced (SQR is a 16-bit application which are especially unfriendly to the WTS environment). As with most Windows products, they presume that one and only one user will run the application at a time. While PeopleSoft does let one configure the application dynamically, the changes are made to global files and registry entries.

Other problems existed – Crystal Reports, when invoked by PeopleSoft using PS/Query, requires registry entries in HKEY_LOCAL_MACHINE (HKLM), which is nominally a "shared" key. While it is possible to create user unique HKLM values in WTS, another solution was chosen. Since the HKLM key had the same value except for the environment (PSENV), the type of the variable was changed from REG_SZ to REG_EXPAND_SZ, which will expand environment variables before returning the string. For example, for a made-up key, the value before the change was

CRYSTAL\BINDIR=N:\HR600\CRYSTAL\BIN

To allow users to share the HKLM registry entries, the value becomes

CRYSTAL\BINDIR=N:\%PSENV%\CRYSTAL\BIN.

2.3 WTS User Interface

WTS (with the MetaFrame add-on) provides one of two methods to access the WTS farm. The first is essentially a desktop in a window on your normal desktop. The other method provides a "seamless" windowed application, where the application appears to be running on your desktop.

The seamless window was chosen for the BSD deployment for a number of reasons. First, the desktop, while it can be restricted somewhat, is a desktop. Not only can it be confusing which desktop one is using, the user can access applications which should properly be restricted. Using published, seamless applications, the WTS administrators decide what is appropriate and "safe." In addition, using a seamless window provides an interface that minimizes end-user training. The users are expecting to see an application window when they (double)click on the PeopleSoft icon. Using published seamless windows preserves, to a large extent, that experience and training.

2.4 Securing MetaFrame

Several departments within SLAC had prior experience with multi-user versions of Windows NT in the form of Ntrigue and WinFrame. It was necessary to retrain

administrators because deploying applications and desktops in the MetaFrame environment was significantly different.

Citrix's Secure ICA client provides from 40-bit to 56-bit DES to 128-bit RC5 encryption of the username and password as well as the data being transmitted and received during the session. This method of access provides the users with a secure tunnel to run the applications to/from the MetaFrame, providing a secured application environment to process sensitive financial and human resource data.

Installations of WTS using the default options do not result in a secure operating environment with respect to registry and file access. A basic method of securing an installation is to replace the access rights of both "guest" and "everyone" group. Replacing these rights with a more tolerable setting of assigned "authenticated users" or even restricting general access to the "domain users" group. This prevents common intrusion techniques and gathering information by anonymously reading system settings via the registry by unauthorized users. The file system should be modified to the most restrictive access possible without impacting the user's ability to run applications on the multi-user host. Since the every user in a multi-user setting must have the "log on locally" right, special attention must be taken with the access permission with the %SYSTEMROOT% (usually c:\winnt), SYSTEM and SYSTEM32 folders. Also due the "log on locally" right required for multi-users, physical security is even more important than in the client/server setting since any domain user may logon to a WTS server at the console.

3. Securing NT Workstations

3.1. Background to Network Install

The open academic environment of SLAC has historically led to departments configuring NT systems more or less as they saw fit, often with little concern towards standardizing the hardware or software or ensuring that systems do not adversely impact the functioning of the network and other systems. Over time, this created enormous management problems, such as tracking down or diagnosing system mis-configurations, timely replacement of failed equipment, and responding to network interruptions. In response to these growing problems, SLAC has worked toward standardization of NT systems by working with Dell to create recommended system bundles, and developing an automated software installation process.

The automated software installation process, or Network Install, began as a way to quickly and easily configure new Dell systems. Using the Network Install, NT systems can be configured in a fraction of the time it takes administrators to configure them manually. The automated process ensures that the operating system and applications are configured per SLAC's standards and brought up to the necessary patch levels, and that the system will work properly on SLAC's network. In an effort to standardize the software on existing, non-Dell NT systems, the Network Install was adapted to work with a variety of other hardware configurations as well. This allows administrators to quickly and easily re-deploy standardized, reliable NT systems when rebuilding older systems or transferring them to new users.

The Network Install is a collection of scripts that utilize automated installation features of Windows NT, and several applications along with a variety of other software tools and utilities. It utilizes a modular construction allowing for easy modification of individual parts as software versions change, drivers are updated or service packs and patches are released. This modular construction makes it easier to update and maintain than a system image-based installation process, such as Ghost, which would require creating new images every time a hardware or software change is necessary. The Network Install is also able to support a much larger variety of hardware than would be feasible with system images. We have also tended to avoid cloning NT installs to preserve the unique machine SID generated during the installation process.

The Network Install consists of five main steps. First, the hard drive is erased, and a new FAT primary partition is created. Second, a script prompts the administrator for configuration information, such as the model and name of the computer, what types of video and Ethernet cards are installed, whether to use DHCP or a specific IP address and gateway, etc. Then, the network card is initialized, the system connects to a share on the network, and begins the installation of NT Workstation. After NT has been installed, the system re-connects to the network share to further configure the operating system and install the standard suite of applications and relevant software updates. Once this process has completed, the administrator need only perform a few final steps, such as installing any non-standard software and setting up printer connections, before the system is ready to be delivered to the user.

3.2. Secure Version of Network Install

In the spirit of SLAC's open environment, NT systems configured with the Network Install are largely default installations in terms of security settings on the registry and file system. Because the level of security required for the new BSDnet is significantly higher than for the rest of the lab, a new, separate version of the Network Install was developed.

The BSDnet Network Install incorporates a number of changes to the system installation and configuration that increase the security of these systems. These changes include:

- Restricting access to system files and directories and sensitive keys in the registry
- Replacing access rights granted to the "Everyone" group with the "Authenticated Users" group
- Removing access rights for the "Guest" group
- Enabling and requiring SMB signing
- Disabling the caching of credentials
- Enabling BIOS passwords
- Removing Dial-up Networking and Remote Access Service components
- Clearing the pagefile at shutdown
- Disabling clear text, LM, and NTLM authentication

Security practices already in place with the existing Network Install, such as requiring NTFS partitions, disabling booting from floppy disks, disabling anonymous connections, requiring complex passwords, renaming the built-in Administrator account and including a legal notice at logon were continued in the BSDnet version of the Network Install. Microsoft Office 97 Service Release 2 (SR-2) and post-SR-2 patches were added, as were patches for Internet Explorer 4.01SP1 and Windows NT 4.0 Service Pack 4 (SP4) and several post-SP4 hotfixes.

Computer Associates' InocuLAN 4.0, and Microsoft Systems Management Server (SMS), Citrix Secure ICA and Security Dynamics ACE client software packages were added to the list of standard software installed by the BSDnet Network Install. InocuLAN was picked to replace the existing anti-virus software because of its greater ability to be centrally managed. The installation of the SMS client greatly enhances the manageability of the BSDnet workstations by allowing administrators to troubleshoot systems remotely and automating the deployment of software updates.

Installation of the ICA client on all BSDnet workstations enables business system users to access PeopleSoft through the Windows Terminal Server farm from any BSDnet workstation, not just from their primary workstation. The Security Dynamics ACE client software is installed but dormant on all BSDnet workstations in preparation for the migration to two-factor token card authentication in the future.

The existing 200 workstations had to be re-installed using this secure Network Install. Most of this was done by a small team of NT administrators during after-office hours when the users were not using their workstations.

3.3 Compatibility with Lab Environment

The addition of the new security enhancements to the Network Install required significant testing and debugging. Requiring SMB signing and disabling all but NTLMv2 authentication initially created connectivity problems with NT servers in other parts of SLAC that had not been configured to enable these security enhancements. The result was that the standards for the rest of the lab were also raised.

Requiring SMB signing does not work with Macintoshes, Linux and other operating systems and services. This was an issue for users who rely on Samba or Transarc AFS client to provide connectivity to UNIX file systems. For these users, one of the SMB signing registry keys (HKEY_LOCAL_MACHINE\SYSTEM \CurrentControlSet\Services\Rdr\Parameters) had to be changed to not required.

Tightening the registry and file system ACL's proved to be particularly complicated, as most applications do not list the minimum access requirements they need in order to function properly. Application failures required numerous adjustments to the file system and registry ACLs, some of which created new failures that had to be fixed as well. Security scans revealed vulnerabilities that had been missed, which then had to be closed. Software standardization of the 'BSDnet' workstations also allowed for conversion from local to roaming profiles, which entailed further testing and troubleshooting.

The experience gained from developing and testing the 'BSDnet' Network Install security enhancements has

enabled the incorporation of some enhancements into the standard Network Install used by the rest of SLAC. This resulted in increasing the security of all of SLAC's NT systems while maintaining an open environment.

4. Other Issues

4.1 Remote Access (RAS/RRAS)

The routers were programmed to accept only Secure ICA connections from the BSD network to the MetaFrame servers. This prevents an unauthorized user from accessing the secured application set from any other segment of the network or the Internet.

For off-site access to the secure application set, users are required to use a 128-bit RC5 PPTP connection to a designated PPTP server, that provides the users with a BSD subnet address which will in turn allow a Secure ICA connection to the MetaFrame farm. PPTP-based access to the BSD subnet is filtering by the router that borders the SLAC network and the Internet. The router only allows the port (TCP 1723) and the underlying protocol (Global Routing Encapsulation (GRE) IP 47) to the BSD PPTP server and one other SLAC PPTP server. All others forms of off-site access have been blocked to add to the security of the BSD network.

4.2 Eliminating Reusable Passwords

In recent years, SLAC and Stanford University has experienced several publicized intrusions. In a few cases, the crackers comprised thousands of user accounts. Therefore, the NT and security staff reviewed the standard practice of reusable passwords in the BSD network. US Department of Energy's Computer Incident Advisory Committee (CIAC, www.ciac.org) has repeatedly stated that reusable passwords are the "weak link" in security at most DOE sites. The solution was to implement a two-phase, non-reusable password authentication system based on Security Dynamics Corporation's ACE/Server.

The use of two-phase authentication involves a user-defined pin code and a computer algorithm generated tokencode. Combining the pin code and the token code makes a single use passcode. Transmitting this passcode over a RC5 encrypted channel and using Microsoft's latest authentication hash, NTLMv2 provides a very high level of security for the username and passcodes. Even if an unauthorized user were to intercept the authentication packets, they would be forced to first decrypt session channel provided by Secure ICA, second decrypt the NTLMv2 hash to reveal the passcode then it would only provide an already expired passcode.

4.3 Network/System Intrusion Detection

In order to identify potential attacks against the BSD and BSD-EPN networks, SLAC is using the ISS RealSecure product to monitor the network for attack signatures. Since these networks are relatively closed, the attack signatures are set at lower thresholds than what might be acceptable on other, more varied, networks. On a system level, ISS System Scanner is used to detect changes to the system, and provide alerts.

5. Conclusions

Using MetaFrame servers meant the PeopleSoft and related applications ran on the NT Servers, not on each client workstation. This allowed for every module within the business application suite to be configured by an administrator. Once configured by the administrator, the application interface was "pre-configured" for the users when they logged into the application servers. Several benefits in configuring applications on MetaFrame:

- Fully encrypted 128-bit RC5 client connection using Citrix's MetaFrame Secure ICA client
- Administrator configured applications that presented a single, well-defined, secure application environment to every user.
- Provided a limited number of systems to monitor for suspicious activity. Since the applications only run on the server farm, only a limited number of NT systems actually contact the database servers over the standard SQL connection (TCP port 1521).
- Ability to "hide" connections by manually selecting a non-standard port(s) to connect the Secure ICA client to the MetaFrame servers. Sniffing standard connection ports, such as NetBIOS over TCP and the published Secure ICA would not provide information to a would-be cracker.

A filtering router blocked all normal access to the PeopleSoft application and direct connections to the database servers via SQL. Only secure shell access and client connections originating from the MetaFrame application servers are now allowed through the router. Any connection attempts through router on ports other than secure shell and Secure ICA are dropped and

logged for later review by the networking and/or security groups. This prevents all but the most hardened security threat from accessing the applications and database deemed secure by this project.

All of the MetaFrame servers were placed in an Extremely Private Network (EPN) with limited access within the EPN and limited access from BSD. All ports that were not used with the secure clients or designated host-to-host connections were blocked.

All systems within the BSD network are Windows NT Workstations 4.0 with a minimum of service pack 4 and several hot fixes. Each system on the BSD network was rebuilt using a specially configured boot floppy process. Each system was erased and loaded with a new; administrator configured version of NT Workstation and a list of standard applications for the desktop. In addition, the Secure ICA client and the Microsoft Point-to-Point Tunneling Protocol (PPTP) client were added to the managed workstations. These secure clients provide 128-bit RC5 encrypted access to the BSD subnet and to the EPN and the MetaFrame servers.

Work supported by Department of Energy contract DE-AC03-76SF00515; SLAC-PUB-8172

Acknowledgements: The work was performed by a collaboration from SLAC's NT Support Group, Security, Networking, Database Support, Web Support, UNIX Systems Group, PeopleSoft Developers and countless "BSDnet" users and SLAC users who gamely or sometimes un-gamely "collaborated" with us. Steve Chadly of Streamline Associates provided invaluable advice and scripts for running PeopleSoft applications in the MetaFrame environment.

SLAC Windows NT Web Site:
http://www2/comp/winnt/winnt.htm

Deployment of Microsoft Windows NT In a Design Engineering Environment.

Jason Sampson, Elwood Coslett, Gary Washington,
Bob Paauwe, Russ Craft, Kevin Wheeler

Intel Corporation

Abstract

This paper details some of the experiences and issues Intel Corporation encountered when developing and deploying a Microsoft Windows NT 4.0* based environment for use in by our chip designers. We are deploying Intel* based Windows NT workstations to engineers designing Intel's next generation products.

We offer our experiences in hope that other groups using NT in a design engineering environment can benefit. Along the way we explain how we solved the problems associated with a tightly controlled design engineering workstation and environment presented us.

1. Introduction

In 1995-1996, Intel Corporation began to deploy Windows NT to design engineering groups at our sites in Folsom, CA, Hillsboro, OR and Haifa, Israel.

All of our design engineering sites have local administrative groups that support the local Design Engineering departments. The groups typically assist with normal day to day operations and maintenance of the environment while working with the Design groups to plan and develop engineering environments to meet their future needs.

The number of design engineers varies from site to site. Design engineering customers total about 5000-6000 individuals world wide, all of which run UNIX and/or NT. The need for a well-planned and functionally complete heterogeneous environment is imperative to the success of each project. The total number NT design engineering workstation number in the 1000-2000 systems worldwide.

2. Workstation Models

Our deployment of NT followed a progression from an PC running as an "Xterm" all the way up to a full-fledged engineering workstation that has full access to both NT and UNIX environments, either natively or through emulation.

2.1 Xterm Model

Our initial effort was to deploy Intel based NT workstations to the engineer's desk. The engineer could access our UNIX environment through an X Windows emulation product. The NT workstations effectively became a powerful X-Term.

By deploying Windows NT, our engineers had the convenience of having all of the productivity tools available on Windows NT (Word*, Excel*, Outlook*, etc) as well as complete access to the UNIX tools needed to design our products. This allowed us to reduce costs by providing a single desktop to meet all of our engineer's needs. It also offers us the ability to move on to an environment where some or most of the design engineer's tasks are done locally to the workstation.

2.2 Standard Infrastructure Release

When our design sites started with NT, they individually developed their own build. This presented problems when projects had to share data and scripts between sites using different builds. A significant amount of work would be necessary to port scripts to another NT environment.

To rectify this problem, we pulled together the best features of the existing NT 4.0 environments and built a standard build, called the Standard Infrastructure Release, for deployment at all of our engineering sites. This standardization not only helps with portability, but also allows us to share the expense of maintaining the environment across multiple sites. This becomes particularly important when we consider our application integration process.

2.3 Work Model

In our design engineering UNIX environments, the design engineers never have administrative (root) rights on their workstation. In addition, applications and data are not permanently stored on the workstation. The UNIX workstations are considered application-less and data-less. This offers the ability to replace systems quickly and have the user back up and running in short time. Should a system fail, there is no reason to worry about re-installing applications, re-enabling daemons, etc. because all workstations have exactly the same build and features enabled. Very rarely is one machine significantly different than another.

In our UNIX based design environments we have developed, and continue to develop, an environment that allows any engineer to run any tool on any workstation anywhere within the company. This "any-any-any" concept is a guiding principle that is also applied to our NT workstations. Paths to applications and tools are consistent across all sites. In addition, the exact same version of each tool or application is usually available at every site.

When developing our NT environments, we worked to apply these same principles. Our NT workstations are application-less and data-less. Also, the exact same version of each tool is available at every site and it is found in same location.

Like with our UNIX workstations, we are able to swap out replacement of defective systems without worry of data loss. The engineers are able to be back to work quickly without having to wait many hours to install all of his or her required applications.

The application-less workstation thus requires that most applications run from a file server. While this is common place in UNIX, it is less commonly found within NT environments. This has forced us to create tools and processes to re-engineer application installations to allow them to run from the network. However, we've also found that some applications just refuse to run from a network location. They usually want to write temp files to the directory they are installed in, instead of using %TEMP%. In that case, we are forced to install the application locally and open the ACLs to allow the application to run.

Since many CAD tools are very large, and the number of tools used by the engineers is extensive, their time is saved re-installing these tools by minimizing the installation process to include only the required local changes like DLLs, registry and profile changes. The bulk of the application is place on the network file server.

In an attempt to apply these effective practices employed to manage our UNIX environments, our NT workstations also have restricted file system permissions. Users generally are not granted administrative rights and can not install just any software package they desire. This places the burden of software installation on the administrative group.

In addition to preventing application installation by the user, we prohibit the permanent storage of data on workstations, thus workstations are not backed up. Personalized components of the application are stored on a personal drive instead of the profile. The data stored on the personal drive, which is a drive located on a network file server, could easily be 50 to 100MB in size. Should we store this in the user's profile, login times would be greatly increased.

Locking down the workstation and preventing users from installing their own application generally increases stability of the workstation. Workstation blue screens and application crashes are reduced.

Another driver for locking down the workstation is our requirement to run batch jobs. To be successful, a batch environment requires a highly reliable and consistent environment. Locking down the workstation and providing consistent tools and tool locations make for a reliable environment.

3. Building the OS Image

Our workstation builds utilize Microsoft's OEM installation process. Using this process, and a few custom scripts, we are able to create an image that is consistent. Any two systems using the same build should not have different components installed, and they are never installed in a different order.

3.1 Partitioning

Our workstations are split into three partitions, C: D: and E:. C: is for OS, system administration tools and the swap file. D: is for any local applications and E: is a user scratch area. By breaking the system into multiple partitions, we're able to reserve space on the system for specific functions. Since the applications are always installed on the D: drive, we can be assured that the C: drive will never fill up because only OS and system administration tools are found on that drive. Same goes for the E: drive, since the user has almost free reign on that drive, we do not have to worry about

an application failing to install because the user has filled the application drive with temporary data.

With each install, the person building the system answers a few questions on the build diskette, including whether to FDISK the system, and then the install process takes over from there.

Partitioning is handled using freeware tools to wipe the partition table and scripted FDISK commands to create the C:, D: and E: partitions. Internally developed command line tools extend the D: and E: partitions beyond the 2GB limit presented by our DOS based build disk.

3.2 Disk Cloning and the OEM Build Process

The OEM build process is an effective means of creating consistent and fairly easy to maintain and extend build images but there are a few limitations.

One of the most serious limitations is that the OEM build process is time consuming; building a system from a boot disk can take a forty-five minutes to a hour depending on network speeds, size and number of applications installed. Our standard build installs the OS, Service Packs, Hotfixes and the productivity applications needed on every system.

To reduce the time taken to build a system, at some sites we've utilized disk cloning to build our systems. The original image is still built using the OEM build process, from which the clone is generated. The result is a workstation image than can be installed in about 10-15 minutes instead of a closer to one hour.

A disadvantage of disk cloning is that it requires a re-creation of the cloned images if you need to change something as simple as a device driver file. We have standardized on only a few workstation models to support, thus the number of images needed is very small. Also the time between releases of our OS image is relatively long, thus we do not have to recreate the cloned images several times. It is our opinion that the time benefits saved installing the cloned image outweighs the cost of creating and recreating the initial clone image.

3.3 ACLs

Once we established a process to install the OS, we extended it to customize the build Microsoft provides. First we used the Zero Admin Kit for Windows* to build our initial ACL settings. This kit, freely available from Microsoft, provides a script to set the ACLs on the local file system to remove Everyone:Read from the file system ACLs and to set them to Users:Read-Only for the C: partition. We applied the same ACL set to the D: partitions, where applications are installed.

Another tweak we made is to %Systemroot%\system32, which has the directory permissions set to Users:Add & Read, Creator-Owner:Full Control. The files provided at build time are set to Users:Read-only. With this setting, users could add new files to the directory, but couldn't change any of the files that were there at build time. The intent of this setting was to allow "good" programs to install to the system, letting them add files to the system32 partition, but not change anything that was already there. This proved in the end to be a very minor impact.

These ACL settings provided the greatest level of controversy with our customers. Many of our customers who were familiar with Microsoft Windows, Windows 95/98 and NT were accustomed to having full administrative rights or the ability in general to install any application they wished. To them, their NT system was their personal computer to do with as they pleased. Our wish was to treat these machines like tightly controlled UNIX workstations, not personal computers. This mindset change was difficult for some to get over, but once they realized their environment was more stable, they began to accept the locked down system.

When these same users began to use our locked down build, they quickly found that most applications could not install on the system. The installation programs would either complain about the inability to copy a DLL, or simply fail with an error message. They would even complain if the version of the file there were copying was exactly the same as what was on the system. The installation required copying the file no matter what. Some good applications would allow you to skip those types of errors and continue with the install, but unfortunately those were very far and few between.

Since the users were not able to install their own applications, it placed the entire burden of application installation on the heads of the administrative support team. We needed a means to install the application on the workstation remotely and consistently.

4. Software Distribution and Installation

The application-less workstation requires that we install as many tools as possible to run from a file server. The file server location also had to be read-only to all users.

We did not want users writing to temp files, etc to the application server.

These requirements resulted in an application installation process that is summarized to updating the local and user's registry, creating short cuts to the network location, and copying DLLs to the system32 directory.

Most of the applications we use do not provide an option of installing and running the application from a network file server. This means in order to make our locked down, application-less workstation model work, we were going to have to re-create the vendor installation that would allow the application to run from the network.

Since we also intended to run batch jobs on our NT workstations at any time of the day, we wanted to minimize job loss by minimizing the number of reboots of a system. Many vendor installs seem to require reboot of the system simply to start a service or perform some other simple function. However, most of the time a reboot was only necessary to update a locked DLL.

We further complicated the problem by requiring that no DLL could ever be downgraded by an application installation. Our experience has been that DLLs generally are backward compatible with an older version. A tool should run, with a high level of certainty, when a newer version of a DLL it depends on is found on the system. Microsoft has published guidelines and made tools available for ISVs to use to perform DLL version checking, but unfortunately, not all applications can go through these checks. Since it is sometimes difficult to tell if the vendor has been able to go through the DLL version checking, we were forced to implement our own DLL versioning system.

4.1 DLL Management

Combining the problem of a lack of network installs for applications with the questionable lack of DLL management in vendor installs, we were forced into creating our installation process and tools.

These internally developed tools perform most of the functions needed to install an application. Where they do not, we rely on NT 4.0 Resource Kit* tools like, SC, to handle specialized tasks.

Our tools also allow the administrator to create an application installation script that would allow the tool to run from a local drive or network location with a few simple variable changes. Every application installation script we make, called a module, is variable-ized. All path locations and site-specific settings are replaced with a variable, and then at installation time, those variables are resolved to values. The result is that one site can create a module specific to their site, and send it to another site, and with just a few changes in a text file, the module should be able to run at the new site.

Perhaps most importantly, our tools nearly give us complete control over what happens to the DLLs on the system. When we started, we used a PERL script to manage our DLLs on the system. When a new application image was created, the DLLs that application installed were checked against a central location. The versions were compared using FILEVER from the NT 4.0 Resource Kit. If the application wanted to install a newer version of the DLL, we would update the central location with that version. A table was kept to track all of the file versions of the DLLs found in the central location.

When an application installed, all of the DLLS provided by the original installation program are copied to the system. Then the PERL script would check whatever was currently on the system against the central location. If the local version were older, we would update the local version with what was found in the central location. If they were the same version, the file was not updated.

When this process was followed, we were able to test multiple applications against the same set of DLLs to validate functionality. However, due to time spans between releases, and the requirement to have site specific tools, the central directory quickly became out-dated.

To improve the DLL management process and to remove dependency on the central location, we built into our installation tool the same logic our PERL script had, excepting that we used native Win32 calls to get the file version. This greatly improved installation time by preventing unnecessary file copies and the execution of a PERL script. It removed the dependency on the central location by preventing the downgrade of the DLL in the first place.

What we found in using this process is that we were nearly able to remove any concern about DLL loading order because the newest version of the DLL should be found on the system. However, what we also found is that locked DLLs are more troublesome than originally thought.

When our installation tool encounters a locked DLL, we use a slightly different process than most vendor

installs use. Most vendor installs set the system up to replace the locked DLL with the newer version at reboot. This forces the user to reboot the system once the application install is completed. The newly installed application cannot use the new DLL until the system reboots, even if the locked DLL becomes unlocked before reboot. Our process renames the locked DLL (e.g. ren msvcrt.dll msvcrt.dll.old23452), copies in the new DLL (e.g. copy msvcrt.dll %Systemroot%\system32), and then use the Win32 API call to delete the locked file at the next reboot. This process allows an application to immediately use the new version of the DLL, should the locked version be released. Thus a reboot is not necessary.

However, DLLs locked by the OS are not released until reboot. This ended up being somewhat problematic.

When querying the file version table of a locked DLL, one that is in use by the OS or other application, we found that the FILEVER tool and the Win32 API calls always return the version of the file loaded into memory. Thus, even if the file has been replaced using our DLL update process, the version that is loaded into memory will be returned. The only way to combat this problem is to require the user to reboot the system before installing additional tools or to rename the locked DLL, check the version, and then rename it back. This of course happens fairly frequently and has made utilization of NT in a batch environment more difficult, since a rebooted system means lost batch jobs.

4.2 Application Installation and Differencing Technology

Our application installation requirements, the ability to run from the network or locally, DLL management, and minimization of reboots, has required us to develop our own tools for software distribution and installation.

As a basis for installs, we are using a process that uses a differencing tool to create a before and after snapshot of the system and then create a new installation script that meets our application requirements.

First we build a bare bones system that includes only the OS, service pack and hotfixes. This is done from an option on our build disk that prevents any of the standard applications from being installed. In effect, we have installed NT using the three boot disks and CD, and running the service pack and hotfix installs manually.

Next we use the differencing tool to take a pre-installation snapshot. The application is installed using the vendor installation program. If we are trying to create a network install, we install it to a simulated network drive. Using SUBST or a local share, we map a pseudo-application drive to X:, our drive letter for our applications. Then the application may also be configured with a customized set of default options. Finally, the differencing tool captures a post-installation snapshot and generates a listing of changes to the workstation that make up that instance of the application installation. We use the results of the differencing tool to generate our installation scripts using our internal tools.

The differencing technology has some distinctive advantages:

- Administrators know exactly what changes are made to the local DLLs, registry and the user's profile. A source of a problem can be tracked down by searching through all of the registry changes, which are kept in plaintext REGEDIT format. A more comprehensive workstation audit is also possible because we know what is supposed to be on the system and where to go to address deficiencies. Without a complete understanding of what files and registry keys are used by an application, problem determination is hampered.
- Conflicts between applications can be identified in advance. Some applications will set registry values to one value, another will want it with another value.
- Removes user interaction from install; installations are consistent. We can create an install that looks exactly the same as every other install.

One problem we've encountered when using this differencing technology is that the output of the tool may be only one possible installation scenario. The differencing tool is not able to identify and replicate the logic built into the install that may be triggered when a particular file or registry entry is present. Thus, the differencing technology is prone to bugs. We've attempted to minimize this problem by trying to guess what the vendor install may do in particular situations, or patch the install when we've encountered a problem with the original install.

In addition, many times registry keys are identified as being changed during the install that either are side effects of the original install or part of normal NT operation, like the Most Recently Used lists or last shutdown time key. Identification of these types of keys is not easy for the new administrator.

The administrator must learn through practice, extensive training and a little bit of art to determine which keys in a registry file are applied and which ones are removed. This learning curve further complicates the application integration process.

Invariably, the resulting application image has bugs or introduces problems through inconsistent application of standards. Many of these issues have been addressed through scripts we've developed to tackle the usual situations an application integrator may encounter, however, intimate knowledge of NT application installation and some good detective work are necessary to provide a high quality application installation.

Clearly the amount of effort needed to integrate an application is very high. In addition, because we force applications to run from locations where they were not tested or intended to run from, and because we are not using the vendor installation method, getting support from a vendor on application issues may be difficult or impossible.

4.3 Split Installs

With most application images, we create two installation scripts. One to apply the system registry and local file system changes, and another script to apply the user profile and personal drive changes. This split, without automation, can also be an intensive, error-prone task.

The system changes are performed by a service running in a local administrative context. Thus any DLLs or registry entries that require administrative rights to update can be handled by this service. The user does not need to have administrative rights to install the product.

The user changes are applied in user context. In our environment the LocalSystem account rarely has rights to access the user's personal drive, so it is necessary to install the user portion of an application by running as that user.

This split of the system and user portions of the application installation also allow us the luxury of being able to "push" the system portion, usually the most resource intensive aspect of an installation, to many systems in the off hours of the day. We are then able to schedule the installation to reduce impact on our file servers and networks. The engineers then simply have to run the user portion of the installation to complete the installation, which is usually very quick and less resource intensive.

We also have the ability to force the installation of the user portion of the application for all logged on users through a user process that is started from the All Users startup group. This user process checks the local system registry for jobs and will execute it.

Our tool suite also offers a pull mechanism. The user can open a GUI program select the application(s) he or she wishes to install and click one button to initiate the install. Depending on the size and location of the application, it will install in just a few minutes, up to a half-hour. Our underlying installation tool works with the GUI to let the user know whether to logoff and log back in, or to reboot the system. If no user action is necessary, the user should be able to immediately open the application and begin using the tool.

5.0 Conclusion

Recently we've taken a hard look at our processes and attempted to re-evaluate their effectiveness or worthiness. We've known all along that our application integration process, including the support for the internally developed tools, was an expensive cost to burden. We felt that the benefits achieved from the policies and processes we implemented outweighed the costs associated with support of the tools and processes we implemented.

Now with the introduction of Microsoft Windows Installer* technology based installation tools, and more ISVs using installation products that allow us to record installation options, many of our higher priority concerns have been addressed. We can now script our application installations, gather a fair amount of information about what changes on the system, and feel comfortable with the DLL management processes. It appears that the industry has made significant strides in addressing many of the issues that had no resolution when we began this process two to three years ago.

With these advances we're close to being able to retire our internally developed tools and rely more on third party solutions for most of our system administration needs.

However, that is not to say that there isn't anything left to fix. A short list of items we would have liked to have seen to make our implementation of NT more successful include:

- More command line tools, which means the ability to script administrative functions. We've had the luxury have having excellent resources internally who can write command line tools, but there are far more important items we should have to worry about.
- We need to minimize reboots. Applications have to be able to use the new version of a DLL without having to reboot.
- The option to install applications locally or on the network. Network and locally installed applications should use the Microsoft specified means to identify areas to write temp files to.
- A means of determining exactly what changes are made during the install both on the filesystem and the registry.

6.0 Acknowledgements

Throughout the past two to three years we've had a number of individuals that have made this paper possible. Thanks go out to the entire iA/NT development team for their hard work and dedication to developing a quality design engineering environment.

7.0 References

MS Windows NT Workstation Deployment Guide - Automating Windows NT Setup. Microsoft Corporation.

Microsoft Zero Administration Kit. Microsoft Corporation.
http://www.microsoft.com/windows/zak/

Microsoft Windows NT Workstation 4.0 Resource kit, Microsoft Corporation, Microsoft Press, 1996.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

SAGE, the System Administrators Guild

The System Administrators Guild, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

Member Benefits:

- Free subscription to *;login:*, the Association's magazine, published eight–ten times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java, Tcl/Tk, and C++, book and software reviews, summaries of sessions at USENIX conferences, and Snitch Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, via the USENIX Online Library on the World Wide Web.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as object-oriented technologies, security, operating systems, electronic commerce, and NT – as many as twelve technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- The right to vote on matters affecting the Association, its bylaws, and election of its directors and officers.
- Discount on BSDI, Inc. products.
- Discount on all publications and software from Prime Time Freeware.
- Savings (10-20%) on selected titles from Academic Press, Morgan Kaufmann, New Riders/Cisco Press/MTP, O'Reilly & Associates, OnWord Press, The Open Group, Sage Science Press, and Wiley Computer Publishing.
- Special subscription rates for Cutter Consortium newsletters, *The Linux Journal, The Perl Journal, IEEE Concurrency, Server/Workstation Expert*, *Sys Admin Magazine*, and all Sage Science Press journals.

Supporting Members of the USENIX Association:

C++ Users Journal
Cirrus Technologies
Cisco Systems, Inc.
CyberSource Corporation
Deer Run Associates
Greenberg News Networks/MedCast Networks
Hewlett-Packard India Software Operations
Internet Security Systems, Inc.
Microsoft Research
MKS, Inc.
Motorola Australia Software Centre
NeoSoft, Inc.
New Riders Press
Nimrod AS
O'Reilly & Associates Inc.
Performance Computing
Questra Consulting
Sendmail, Inc.
Server/Workstation Expert
TeamQuest Corporation
UUNET Technologies, Inc.
Windows NT Systems Magazine
WITSEC, Inc.

Sage Supporting Members:

Atlantic Systems Group
Collective Technologies
D. E. Shaw & Co.
Deer Run Associates
Electric Lightwave, Inc.
ESM Services, Inc.
GNAC, Inc.
Mentor Graphics Corp.
Microsoft Research
MindSource Software Engineers
Motorola Australia Software Centre
New Riders Press
O'Reilly & Associates Inc.
Remedy Corporation
RIPE NCC
SysAdmin Magazine
Taos Mountain
TransQuest Technologies, Inc.
Unix Guru Universe

For further information about membership, conferences or publications, contact:
USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA.
Phone: 510-528-8649. Fax: 510-548-5738.
Email: *office@usenix.org.*
URL: *http://www.usenix.org.*

ISBN 1-880446-30-8